

A Deterministic Almost-Tight Distributed Algorithm for Approximating Single-Source Shortest Paths*

Monika Henzinger[†] Sebastian Krinninger[‡] Danupon Nanongkai[§]

Abstract

We present a deterministic $(1 + o(1))$ -approximation $O(n^{1/2+o(1)} + D^{1+o(1)})$ -time algorithm for solving the single-source shortest paths problem on distributed weighted networks (the **CONGEST** model); here n is the number of nodes in the network and D is its (hop) diameter. This is the first non-trivial deterministic algorithm for this problem. It also improves (i) the running time of the randomized $(1 + o(1))$ -approximation $\tilde{O}(n^{1/2}D^{1/4} + D)$ -time¹ algorithm of Nanongkai [STOC 2014] by a factor of as large as $n^{1/8}$, and (ii) the $O(\epsilon^{-1} \log \epsilon^{-1})$ -approximation factor of Lenzen and Patt-Shamir’s $\tilde{O}(n^{1/2+\epsilon} + D)$ -time algorithm [STOC 2013] within the same running time. Our running time matches the known time lower bound of $\Omega(n^{1/2}/\log n + D)$ [Das Sarma et al. STOC 2011] modulo some lower-order terms, thus essentially settling the status of this problem which was raised at least a decade ago [Elkin SIGACT News 2004]. It also implies a $(2 + o(1))$ -approximation $O(n^{1/2+o(1)} + D^{1+o(1)})$ -time algorithm for approximating a network’s weighted diameter which almost matches the lower bound by Holzer et al. [PODC 2012].

In achieving this result, we develop two techniques which might be of independent interest and useful in other settings: (i) a deterministic process that replaces the “hitting set argument” commonly used for shortest paths computation in various settings, and (ii) a simple, deterministic, construction of an $(n^{o(1)}, o(1))$ -hop set of size $O(n^{1+o(1)})$. We combine these techniques with many distributed algorithmic techniques, some of which from problems that are not directly related to shortest paths, e.g. ruling sets [Goldberg et al. STOC 1987], source detection [Lenzen, Peleg PODC 2013], and partial distance estimation [Lenzen, Patt-Shamir PODC 2015]. Our hop set construction also leads to single-source shortest paths algorithms in two other settings: (i) a $(1 + o(1))$ -approximation $O(n^{o(1)})$ -time algorithm on *congested cliques*, and (ii) a $(1 + o(1))$ -approximation $O(n^{o(1)} \log W)$ -pass $O(n^{1+o(1)} \log W)$ -space *streaming* algorithm, when edge weights are in $\{1, 2, \dots, W\}$. The first result answers an open problem in [Nanongkai, STOC 2014]. The second result partially answers an open problem raised by McGregor in 2006 [sublinear.info, Problem 14].

*A preliminary version of this paper appears in Symposium on Theory of Computing (STOC) 2016.

[†]University of Vienna, Faculty of Computer Science, Austria. The research leading to this work has received funding from the European Union’s Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 317532 and from the European Research Council under the European Union’s Seventh Framework Programme (FP7/2007-2013)/ERC Grant Agreement number 340506. This work was done in part while visiting the Simons Institute for Theory of Computing.

[‡]Max Planck Institute for Informatics, Saarland Informatics Campus, Germany. This work was done while at the University of Vienna, Austria, and in part while visiting the Simons Institute for the Theory of Computing.

[§]KTH Royal Institute of Technology, Sweden. Support by Swedish Research Council grant 2015-04659 “Algorithms and Complexity for Dynamic Graph Problems”

¹ \tilde{O} hides polylogarithmic factors.

Contents

1	Introduction	3
2	Preliminaries	7
2.1	Notation	7
2.2	The CONGEST Model and the Problem	7
2.3	Toolkit	8
3	Deterministic Hop Set Construction	10
3.1	Deterministic Clusters	11
3.1.1	Computing Priorities \mathcal{A}	11
3.1.2	Computing Clusters	13
3.2	Hop Reduction with Additive Error	13
3.3	Hop Reduction without Additive Error	20
3.4	Computing the Hop Set	21
4	Distributed Single-Source Shortest Paths Algorithm on Networks with Arbitrary Topology	23
4.1	Computing an Overlay Network Deterministically	23
4.1.1	Types of Nodes	25
4.1.2	Selecting Centers via Ruling Sets	26
4.1.3	Computing Distances to Centers	27
4.1.4	Completing the Proof of Theorem 4.1	27
4.2	Computing a Hop Set on an Overlay Network	28
4.2.1	Computing Bounded-Distance Single-Source Shortest Paths	28
4.2.2	Computing Priorities	29
4.2.3	Computing Clusters	30
4.2.4	Computing the Hop Reduction with Additive Error	31
4.2.5	Computing the Hop Reduction without Additive Error	31
4.2.6	Computing the Hop Set	31
4.2.7	Routing via the Hop Set	32
4.3	Final Steps	32
5	Algorithms on Other Settings	33
5.1	Congested Clique	33
5.2	Streaming Algorithm	34
6	Conclusion and Open Problems	34
	References	35
A	Proof of Lemma 2.2	42
B	Proof of Lemma 3.1	42
C	Ruling Set Algorithm	43

1 Introduction

In the area of *distributed graph algorithms* we study the complexity for a network to compute its own topological properties, such as minimum spanning tree, maximum matching, or distances between nodes. A fundamental question in this area that has been studied for many years is how much *time complexity* is needed to solve a problem in the so-called *CONGEST model* (e.g. [GKP98, PR00, Elk06, DHK⁺12, LPS13]). In this model (see Section 2 for details), a network is modeled by a weighted undirected graph G , where each node represents a processor that initially only knows its adjacent edges and their weight, and nodes must communicate with each other to discover global topological properties of the network. The communication between nodes is carried out in *rounds*, where in each round each node can send a small message to each neighbor. The time complexity is measured as the number of rounds needed to finish the task. It is usually measured by n , the number of nodes in the network, and D , the diameter of the communication network (when edge weights are omitted). Typically, $D \ll n$.

In this paper, we consider the problem of *approximating single-source shortest paths* (SSSP). In this problem, a node s is marked as the *source node*, and the goal is for every node to know how far it is from s . The unweighted version – the *breadth-first search tree* computation – is one of the most basic tools in distributed computing, and is well known to require $\Theta(D)$ time (e.g. [Pel00]). In contrast, the only available solution for the weighted case is the distributed version of the Bellman-Ford algorithm [Bel58, For56], which requires $O(n)$ time to compute an exact solution. In 2004, Elkin [Elk04] raised the question whether distributed *approximation* algorithms can help improving this time complexity and showed that any α -approximation algorithm requires $\Omega((n/\alpha)^{1/2}/\log n + D)$ time [Elk06]. Das Sarma et al. [DHK⁺12] (building on [PR00, KKP13]) later strengthened this lower bound by showing that any poly(n)-approximation (randomized) algorithm requires $\Omega(n^{1/2}/\log n + D)$ time. This lower bound was later shown to hold even for quantum algorithms [EKN⁺14].

Since running times of the form² $\tilde{O}(n^{1/2} + D)$ show up in many distributed algorithms (e.g. MST [KP98, PR00], connectivity [Thu97, PT11], and minimum cut [NS14, GK13]) it is natural to ask whether the lower bound of [DHK⁺12] can be matched. The first answer to this question is a randomized $O(\epsilon^{-1} \log \epsilon^{-1})$ -approximation $\tilde{O}(n^{1/2+\epsilon} + D)$ -time algorithm by Lenzen and Patt-Shamir [LPS13]³. The running time of this algorithm is nearly tight if we are satisfied with a large approximation ratio. For a small approximation ratio, Nanongkai [Nan14] presented a randomized $(1 + o(1))$ -approximation $\tilde{O}(n^{1/2}D^{1/4} + D)$ -time algorithm. The running time of this algorithm is nearly tight when D is small, but can be close to $\tilde{\Theta}(n^{2/3})$ even when $D = o(n^{2/3})$. This created a rather unsatisfying situation: First, one has to sacrifice a large approximation factor in order to achieve the near-optimal running time, and to achieve a $(1 + o(1))$ approximation factor, one must pay an additional running time of $D^{1/4}$ which could be as far from the lower bound as $n^{1/8}$ when D is large. Because of this, the question whether we can close the gap between upper and lower bounds for the running time of $(1 + o(1))$ -approximation algorithms was left as the main open problem in [Nan14, Problem 7.1]. Secondly, and more importantly, both these algorithms are randomized.

²Throughout, we use \tilde{O} to hide polylogarithmic factors.

³Note that the result of Lenzen and Patt-Shamir in fact solves a more general problem.

Given that designing deterministic algorithms is an important issue in distributed computing. This leaves an important open problem whether there is a deterministic algorithm that is faster than Bellman-Ford’s algorithm, i.e. that runs in *sublinear-time*.

Our Results. In this paper, we resolve the two issues above. We present a deterministic $(1 + o(1))$ -approximation $O(n^{1/2+o(1)} + D^{1+o(1)})$ -time algorithm for this problem (the $o(1)$ term in the approximation ratio hides a $1/\text{polylog } n$ factor and the $o(1)$ term in the running time hides an $O(\sqrt{\log \log n / \log n})$ factor). Our algorithm almost settles the status of this problem as its running time matches the lower bound of Das Sarma et al. up to an $O(n^{o(1)})$ factor.

Since an α -approximate solution to SSSP gives a 2α -approximate value of the network’s *weighted diameter* (cf. Section 2), our algorithm can $(2 + o(1))$ -approximate the weighted diameter within the same running time. Previously, Holzer et al. [HW12] showed that for any $\epsilon > 0$, a $(2 - \epsilon)$ -approximation algorithm for this problem requires $\tilde{\Omega}(n)$ time. Thus, the approximation ratio provided by our algorithm cannot be significantly improved without increasing the running time. The running time of our algorithm also cannot be significantly improved because of the lower bound of $\Omega(n^{1/2} / \log n + D)$ [DHK⁺12] for approximate SSSP which holds for any $\text{poly}(n)$ -approximation algorithm.

Using the same techniques, we also obtain a deterministic $(1 + o(1))$ -approximation $O(n^{o(1)})$ -time algorithm for the special case of *congested clique*, where the underlying network is fully-connected. This gives a positive answer to Problem 7.5 in [Nan14]. Previous algorithms solved this problem exactly in time $\tilde{O}(n^{1/2})$ [Nan14] and $\tilde{O}(n^{1/3})$ [CKK⁺15], respectively, and $(1 + o(1))$ -approximately in time $O(n^{0.158})$ [CKK⁺15]⁴. We can also compute a $(2 + o(1))$ -approximation of the weighted diameter within the same running time. Holzer and Pinski proved that computing a $(2 - o(1))$ -approximation of the diameter requires $\tilde{\Omega}(n)$ time in the worst case [HP15] in the congested clique.

Our techniques also lead to a (non-distributed) *streaming algorithm* for $(1 + o(1))$ -approximate SSSP where the edges are presented in an arbitrary-order stream, and an algorithm with limited space (preferably $\tilde{O}(n \log W)$, when edge weights are in $\{1, 2, \dots, W\}$) reads the stream in *passes* to determine the answer (see, e.g., [McG14] for a recent survey). It was known that $\tilde{O}(n \log W)$ space and one pass are enough to compute an $O(\log n / \log \log n)$ -spanner and therefore approximate all distances up to a factor of $O(\log n / \log \log n)$ [FKM⁺08] (see also [FKM⁺05, Bas08, EZ06, Elk11]). This almost matches a lower bound which holds even for the s - t -shortest path problem (stSP), where we just want to compute the distance between two specific nodes s and t [FKM⁺08]. On unweighted graphs one can compute $(1 + \epsilon, \beta)$ -spanners in β passes and $O(n^{1+1/k})$ space [EZ06], and get $(1 + \epsilon)$ -approximate SSSP in a total of $O(\beta/\epsilon)$ passes. In 2006, McGregor raised the question whether we can solve stSP better with a larger number of passes (see [Sub]). Very recently Guruswami and Onak [GO16] showed that a p -pass algorithm on unweighted graphs requires $\tilde{\Omega}(n^{1+\Omega(1/p)} / O(p))$ space. This does not rule out, for example, an $O(\log n)$ -pass $\tilde{O}(n)$ -space algorithm. Our algorithm, which solves the more general SSSP problem, gets close to this: it takes $O(n^{o(1)} \log W)$ passes and $O(n^{1+o(1)} \log W)$ space.

⁴With this running time, [CKK⁺15] can in fact solve the all-pairs shortest paths problem. Also see [LG16] for further developments in the direction of [CKK⁺15].

Overview of Techniques. Our crucial new technique is a deterministic process that can replace the following “path hitting” argument: For any c , if we pick $\tilde{\Theta}(c)$ nodes uniformly at random as *centers* (typically $c = n^{1/2}$), then a shortest path containing n/c edges will contain a center with high probability. This allows us to create shortcuts between centers – where we replace each path of length n/c between centers by an edge of the same length – and focus on computing shortest paths between centers. This argument has been repetitively used to solve shortest paths problems in various settings (e.g. [UY91, HK95, DI06, BHS07, RZ11, San05, DFI05, DFR09, Mađ10, Ber16, LPS13, Nan14]). In the sequential model a set of centers of size $\tilde{\Theta}(c)$ can be found deterministically with the greedy hitting set heuristic once the shortest paths containing n/c edges are known [Zwi02, Kin99]. We are not aware of any non-trivial deterministic process that can achieve the same effect in the distributed setting.⁵

In this paper, we develop a new deterministic process to pick $\tilde{\Theta}(c)$ centers. The key new idea is to carefully divide nodes into $\tilde{O}(1)$ *types*. Roughly speaking, we associate each type t with a value w_t and make sure that the following properties hold: (i) every path π with $\Omega(n/c)$ edges and weight $\Theta(w_t)$ contains a node of type t , and (ii) there is a set of $O(n/c)$ *centers of type t* such that every node of type t has at least one center at distance $o(w_t)$. We define the set of centers to be the collection of centers of all types. The two properties together guarantee that every long path will be *almost* hit by a center: for every path π containing at least n/c edges, there is a center whose distance to some node in π is $o(w(\pi))$ where, $w(\pi)$ is the total weight of π . This is already sufficient for us to focus on computing shortest paths only between centers as we would have done after picking centers using the path hitting argument. To the best of our knowledge, such a deterministically constructed set of centers that almost hits *every* long path was not known to exist before. The process itself is not constrained to the distributed setting and thus might be useful for derandomizing other algorithms that use the path hitting argument.

To implement the above process in the distributed setting, we use the *source detection* algorithm of Lenzen and Peleg [LP13] to compute the type of each node. We then use the classic *ruling set* algorithm of Goldberg et al. [GPS88] to compute the set of centers of each type that satisfies the second property above. (A technical note: we also need to compute a bounded-depth shortest-path tree from every center. In [Nan14], this was done using the *random delay* technique. We also derandomize this step by adapting the *partial distance estimation algorithm* of Lenzen and Patt-Shamir [LP15].)

Another tool, which is the key to the improved running time, is a new *hop set* construction. An (h, ϵ) -hop set of a graph $G = (V, E)$ is a set F of weighted edges such that the distance between any pair of nodes in G can be $(1 + \epsilon)$ -approximated by their h -hop distance (given by a path containing at most h edges) on $G' = (V, E \cup F)$ (see Section 2 for details). The notion of hop set was defined by Cohen [Coh00] in the context of parallel computing, although it has been used implicitly earlier, e.g. [UY91, KS92] (see [Coh00] for a detailed discussion). The previous SSSP algorithm [Nan14] was able to construct an $(n/k, 0)$ -hop set of size kn ,

⁵We note that the algorithm of King [Kin99] for constructing a blocker can be viewed as an efficient way to greedily pick a hitting set by efficiently computing the scores of nodes. The process is as highly sequential like other greedy heuristics.

for any integer $k \geq 1$, as a subroutine (in [Nan14] this was called *shortest paths diameter reduction*⁶). In this paper, we show that this subroutine can be replaced by the construction of an $(n^{o(1)}, o(1))$ -hop set of size $O(n^{1+o(1)})$.

Our hop set construction is based on computing *clusters* which is the basic subroutine of Thorup and Zwick’s distance oracles [TZ05] and spanners [TZ05, TZ06]. It builds on a line of work in dynamic graph algorithms. In [Ber09], Bernstein showed that clusters can be used to construct an $(n^{o(1)}, o(1))$ -hop set of size $O(n^{1+o(1)})$. Later in [HKN14], we showed that the same kind of hop set can be constructed by using a structure similar to clusters while restricting the shortest-path trees involved to some small distance and use such a construction in the dynamic (more precisely, decremental) setting. The construction is, however, fairly complicated and heavily relies on randomization. In this paper, we build on the same idea, i.e., we construct a hop set using bounded-distance clusters. However, our construction is significantly simplified, to the point that we can treat the cluster computation as a black box. This makes it easy to apply on distributed networks and to derandomize. To this end, we derandomize the construction simply by invoking the deterministic clusters construction of Roditty, Thorup, and Zwick [RTZ05] and observe that it can be implemented efficiently on distributed networks⁷. We note that it might be possible to use Cohen’s hop set construction instead. However, Cohen’s construction heavily relies on randomness and derandomizing it seems significantly more difficult.

Updates. After the preliminary version of this paper appeared ([HKN16]), [BKK⁺16] showed that the $n^{o(1)}$ term in our bounds can be eliminated. Elkin and Neiman showed the first construction of sparse hop sets with a constant number of hops [EN16a], and showed an application in computing approximate shortest paths from s sources. In particular, using our hop set and a modification of the framework in [Nan14] and this paper⁸, this problem can be solved in $O((sn)^{1/2+o(1)} + D^{o(1)})$ rounds. Elkin and Neiman showed a hop set which can be used to reduce the bound to $\tilde{O}((sn)^{1/2} + D)$ when $s = n^{\Omega(1)}$. In [EN16b], they also showed further applications of hop set in the distributed construction of routing schemes. It was pointed out by Patt-Shamir (see [Tse15]) that using our algorithm as a black box, one can simplify and obtain improved running time in the construction of compact routing tables in [LP15]. (On the other hand, we note that our construction is based on many ideas from [LP15].) Our hop set construction also found applications in metric-tree embeddings [FL16].

Organization. We start by introducing notation and the main definition in Section 2. Then in Section 3 we explain the deterministic hop set construction in, which is based on a variation of Thorup and Zwick’s clusters [TZ05]. In Section 4, we give our main result, namely the $(1 + o(1))$ -approximation $O(n^{1/2+o(1)} + D^{1+o(1)})$ -time algorithm. In that section we explain the deterministic process for selecting centers mentioned above, as well as how to implement the hop set construction in the distributed setting. Finally, our remaining results are proved in Section 5.

⁶This follows the notion of shortest paths diameter used earlier in distributed computing [KP08]

⁷We note that Thorup-Zwick’s distance oracles and spanners were considered before in the distributed setting (e.g. [LP15, DSDP15])

⁸More precisely, following Elkin and Neiman [EN16a], one constructs an overlay network of size \sqrt{sn} instead of \sqrt{n} as done in this paper.

2 Preliminaries

2.1 Notation

In this paper we consider weighted undirected graphs. For a set of edges E , the weight of each edge $(u, v) \in E$ is given by a function $w(u, v, E)$. If $(u, v) \notin E$, we set $w(u, v, E) = \infty$. For a graph $G = (V, E)$, we define $w(u, v, G) = w(u, v, E)$. Whenever we define a set of edges E as the union of two sets of edges $E_1 \cup E_2$, we set the weight of every edge $(u, v) \in E$ to $w(u, v, E) = \min(w(u, v, E_1), w(u, v, E_2))$. We denote the weight of a path π in a graph G by $w(\pi, G)$ and the number of edges of π by $|\pi|$.

Given a graph $G = (V, E)$ and a set of edges $F \subseteq V^2$, we define $G \cup F$ as the graph that has V as its set of nodes and $E \cup F$ as its set of edges. The weight of each edge (u, v) is given by $w(u, v, G \cup F) = w(u, v, E \cup F) = \min(w(u, v, E), w(u, v, F))$.

We denote the distance between two nodes u and v , i.e., the weight of the shortest path between u and v , by $d(u, v, G)$. We define the distance between a node u and a set of nodes $A \subseteq V$ by $d(u, A, G) = \min_{v \in A} d(u, v, G)$. For every pair of nodes u and v we define distance up to range R by

$$d(u, v, R, G) = \begin{cases} d(u, v, G) & \text{if } d(u, v, G) \leq R \\ \infty & \text{otherwise.} \end{cases}$$

and for a node v and set of nodes $A \subseteq V$ by $d(u, A, R, G) = \min_{v \in A} d(u, v, R, G)$.

For any positive integer h and any nodes u and v , we define the *bounded-hop distance* between u and v , denoted by $d^h(u, v, G)$, as the weight of the shortest among all u - v paths containing at most h edges. More precisely, let $\Pi^h(u, v)$ be the set of all paths between u and v such that each path $\pi \in \Pi^h(u, v)$ contains at most h edges. Then, $d^h(u, v, G) = \min_{\pi \in \Pi^h(u, v)} w(\pi, G)$ if $\Pi^h(u, v) \neq \emptyset$, and $d^h(u, v, G) = \infty$ otherwise.

We denote the hop-distance between two nodes u and v , i.e., the distance between u and v when we treat G as an unweighted graph, by $\text{hop}(u, v, G)$. The *hop diameter* of graph G is defined as $D(G) = \max_{u, v \in V(G)} \text{hop}(u, v, G)$. When G is clear from the context, we use D instead of $D(G)$. We note that this is different from the *weighted diameter*, which is defined as $WD(G) = \max_{u, v \in V(G)} d(u, v, G)$. Throughout this paper we use “diameter” to refer to the hop diameter (as it is typically done in the literature).

Given any graph $G = (V, E)$, any integer h , and $\epsilon \geq 0$, we say that a set of weighted edges F is an (h, ϵ) -hop set of G if

$$d(u, v, G) \leq d^h(u, v, H) \leq (1 + \epsilon)d(u, v, G),$$

where $H = (V, E \cup F)$. In this paper we are only interested in $(n^{o(1)}, o(1))$ -hop sets of size $O(n^{1+o(1)})$. We refer to them simply as “hop sets” (without specifying parameters).

2.2 The CONGEST Model and the Problem

A network of processors is modeled by an undirected weighted n -node m -edge graph G , where nodes model the processors and edges model the *bounded-bandwidth* links between the processors. Nodes are assumed to have unique IDs in the range $\{1, 2, \dots, \text{poly}(n)\}$ and infinite computational power. We denote by λ the number of bits used to represent each ID, i.e., $\lambda = O(\log n)$. Each node has limited topological knowledge; in particular, every node u

only knows the IDs of each neighbor v and $w(u, v, G)$. As in [Nan14], we assume that edge weights are integers in $\{1, 2, \dots, W\}$, and $W = \text{poly log } n$. This is a typical assumption as it allows to encode the weight of an edge in one (or a constant number of) messages.

The distributed communication is performed in *rounds*. At the beginning of each round, all nodes wake up simultaneously. Each node u then sends an arbitrary message of $B = O(\log n)$ bits through each edge (u, v) , and the message will arrive at node v at the end of the round. The *running time* of a distributed algorithm is the worst-case number of rounds needed to finish a task. It is typically analyzed based on n (the number of nodes) and D (the network diameter).

Definition 2.1 (Single-Source Shortest Paths (SSSP)). *In the single-source shortest paths problem (SSSP), we are given a weighted network G and a source node s (the ID of s is known to every node). We want to find the distance between s and every node v in G , denoted by $d(s, v, G)$. In particular, we want every node v to know the value of $d(s, v, G)$.*

Recovering shortest paths. We note that although we define the problem to be computing the distances, we can easily recover the shortest paths in the sense that every node u knows its neighbor v that is in the shortest path between u and s . This is because our algorithm computes the distance approximation that satisfies the following property

$$\text{every node } u \neq s \text{ has a neighbor } v \text{ such that } d'(s, v) + w(u, v, G) \leq d'(s, u), \quad (1)$$

where $d'(s, v)$ is the approximate distance between s and v . For any distance approximation d' that satisfies Equation (1), we can recover the shortest paths by assigning v as the intermediate neighbor of u in the shortest paths between u and s .

It can be easily checked throughout that the approximate distance function that we compute satisfies Equation (1). This is simply because our algorithm always rounds an edge weight $w(u, v, G)$ up to some weight $w'(u, v)$, and computes the approximate distances based on this rounded edge weight. For this reason, we can focus only on computing the approximate distances in this paper.

2.3 Toolkit

In the following we review known results used for designing our algorithm. The first is a weight-rounding technique [Coh98, Zwi02, Ber09, Mađ10, Ber16, Nan14] that we repeatedly use to scale down edge weights at the cost of approximation. As we will use this technique repeatedly, we give a proof in Appendix A for completeness

Lemma 2.2 ([Nan14]). *Let $h \geq 1$ and let G be a graph with integer edge weights from 1 to W . For every integer $0 \leq i \leq \lfloor nW \rfloor$, set $\rho_i = \frac{\epsilon 2^i}{h}$ and let G_i be the graph with the same nodes and edges as G and weight $w(u, v, G_i) = \lceil \frac{w(u, v, G)}{\rho_i} \rceil$ for every edge (u, v) . Then for all pairs of nodes u and v and every $0 \leq i \leq \lfloor nW \rfloor$*

$$\rho_i \cdot d(u, v, G_i) \geq d(u, v, G). \quad (2)$$

Moreover, if $2^i \leq d^h(u, v, G) \leq 2^{i+1}$, then

$$d(u, v, G_i) \leq (1 + 2/\epsilon)h \quad \text{and} \quad (3)$$

$$\rho_i \cdot d(u, v, G_i) \leq (1 + \epsilon)d^h(u, v, G). \quad (4)$$

An important subroutine in our algorithm is a procedure for solving the *source detection* problem [LP13] in which we want to find the σ nearest “sources” in a set S for every node u , given that they are of distance at most γ from u . Ties are broken lexicographically. The following definition is from [LP15]

Definition 2.3 ((S, γ, σ) -detection). *Consider a graph G , a set of “sources” $S \subseteq V(G)$, and parameters $\gamma, \sigma \in \mathbb{N}$. For any node u let $L(u, S, \gamma, \sigma, G)$ denote the list resulting from ordering the set $\{(d(u, v, G), v) \mid v \in S \wedge d(u, v, G) \leq \gamma\}$ lexicographically in ascending order, i.e., where*

$$\begin{aligned} ((d(u, v, G), v) < (d(u, v', G), v')) &\iff \\ (d(u, v, G) < d(u, v', G)) \vee (d(u, v, G) = d(u, v', G) \wedge v < v'), \end{aligned}$$

and restricting the resulting list to the first σ entries. The goal of the (S, γ, σ) -detection problem is to compute $L(u, S, \gamma, \sigma, G)$ for every node $u \in V(G)$. In the distributed setting we assume that each node knows γ, σ , and whether it is in S or not and the goal is that every node $u \in V(G)$ knows its list $L(u, S, \gamma, \sigma, G)$,

Lenzen and Peleg designed a source detection algorithm for unweighted networks [LP13]. One can also run the algorithm on weighted networks, following [LP15, proof of Theorem 3.3], by simulating each edge of some weight L with an unweighted path of length L . Note that nodes in the paths added in this way are never sources.

Theorem 2.4 ([LP13]). *In the CONGEST model there is an algorithm for solving the (S, γ, σ) -detection problem in $\min(\gamma, WD) + \min(\sigma, |S|)$ rounds on weighted networks, where WD is the weighted diameter.*

We also use another source detection algorithm: Roditty, Thorup, and Zwick [RTZ05] also solve a variant of the source detection problem with $\gamma = \infty$ in their centralized algorithm for computing distances oracles and spanners deterministically. They reduce the source detection problem to a sequence of single-source shortest paths computations on graphs with some additional nodes and edges. Their algorithm can easily be generalized to arbitrary γ .

Theorem 2.5 (implicit in [RTZ05]). *In the sequential model, the (S, γ, σ) -detection problem in directed graphs with positive edge weights can be solved by performing σ single-source shortest paths computations up to distance γ on graphs with at most $O(n)$ nodes and $O(m)$ edges.*

Here we give a short sketch of the algorithm. The algorithm works in phases, in the j -th phase it finds for every node $v \in V$ the j -th source node of S of distance at most γ (with ties broken arbitrarily). In the first phase we find for each node $v \in V$ its closest source node of distance at most γ as follows: We add an artificial super-source s^* to G with a 0-length edge from every node in S to s^* . Then we perform a single-source shortest path computation with source s^* up to distance at most γ , resulting in a shortest-path tree rooted at s^* . For each node $v \in V$ its ancestor in this tree that is a child of s^* is its closest source. We use $U_j(v)$ to denote the (up to) j closest sources for v that the algorithm found in the first j phases for $1 \leq j \leq \sigma$. In phase $j > 1$ we construct a graph G_j with the following node set: We add to V a new super-source node s^* and a copy \bar{u} of every node $u \in A_i$. The

graph G_j contains the following edges: For each new node \bar{u} it contains a 0-length edge to s^* . Additionally for each edge $(y, v) \in E$, we add the edge (with length $w(y, v, G)$) to G_j if $U_{j-1}(v) = U_{j-1}(y)$ and otherwise we add the edge (y, \bar{u}) with length $w(y, v, G) + d(v, u, G)$, where u is the source in $U_{j-1}(v) \setminus U_{j-1}(y)$ with smallest distance from v (with ties broken lexicographically). This edge is a “shortcut” that replaces all possible paths from y through the edge (y, v) to a source not in $U_{j-1}(y)$ by an edge to the node \bar{u} representing the closest such source u . Then we perform a single-source shortest path computation with source s^* up to distance at most γ in G_j . The j -th closest source for each node v is its ancestor that is a child of s^* in the resulting shortest-path tree. Note that each graph G_j consists of at most $n + |S| + 1$ nodes and $m + |S|$ edges and, as there are σ phases, the algorithm requires σ single-source shortest path computations.

Another subproblem arising in our algorithm is the computation of *ruling sets*. The following definition was adapted from the recent survey of Barenboim and Elkin [BE13, Section 9.2].

Definition 2.6 (Ruling Set). *For a (possibly weighted) graph G , a subset $U \subseteq V(G)$ of nodes, and a pair of positive integers α and β , a set $T \subset U$ is an (α, β) -ruling set for U in G if*

1. *for every pair of distinct nodes $u, v \in T$, it holds that $d(u, v, G) \geq \alpha$, and*
2. *for every node $u \in U \setminus T$, there exists a “ruling” node $v \in T$, such that $d(u, v, G) \leq \beta$.*

The classic result of Goldberg et al. [GPS88] shows that in the distributed setting, for any $c \geq 1$, we can compute a $(c, c\lambda)$ -ruling set deterministically in $O(c \log n)$ rounds, where λ is the number of bits used to represent each ID in the network. Since it was not explicitly stated that this algorithm works in the CONGEST model, we sketch an implementation of this algorithm in Appendix C (see [BE13, Chapter 9.2] and [Pel00, Chapter 22] for more detailed algorithm and analysis).

Theorem 2.7 (implicit in [GPS88]). *In the CONGEST model there is an algorithm that, for every $c \geq 1$, computes a $(c, c\lambda)$ -ruling set in $O(c \log n)$ rounds, where λ is the number of bits used to represent each ID in the network.*

3 Deterministic Hop Set Construction

In this section we present an deterministic algorithm for constructing an $(n^{o(1)}, o(1))$ -hop set. In Section 3.2 we give an algorithm that computes a set of edges F that reduces the number of hops between all pairs of nodes in the following way for some fixed $\Delta \geq 1$: if the shortest path has weight R , then using the edges of F , we can find a path with $\tilde{O}(R/\Delta)$ edges at the cost of a multiplicative error of $o(1)$ and an additive error of $n^{o(1)}\Delta$. Our algorithm obtains F by computing the *clusters* of the graph. We explain clusters and their computation in Section 3.1. In Section 3.3, we show how to repeatedly apply the first algorithm for different edge weight modifications to obtain a set of edges F providing the following stronger hop reduction for all pairs of nodes: if the shortest path has h hops, then, using the edges of F , we can find a path with $\tilde{O}(h/\Delta)$ hops at the cost of a multiplicative error of $o(1)$ and no additive error. Finally, in Section 3.4, we obtain the hop set by repeatedly applying the hop reduction.

3.1 Deterministic Clusters

The basis of our hop set construction is a structure called *cluster* introduced by Thorup and Zwick [TZ05] who used it, e.g., to construct distance oracles [TZ05] and spanners [TZ06] of small size. Consider an integer p such that $2 \leq p \leq \log n$ and a hierarchy of sets of nodes $(A_i)_{0 \leq i \leq p}$ such that $A_0 = V$, $A_p = \emptyset$, and $A_0 \supseteq A_1 \supseteq \dots \supseteq A_p$. We say that a node v has *priority* i if $v \in A_i \setminus A_{i+1}$ (for $0 \leq i \leq p-1$). For every node $v \in V$ the *cluster* of v in G is defined as

$$C(v, \mathcal{A}, G) = \{u \in V \mid d(u, v, G) < d(u, A_{i+1}, G)\},$$

where i is the priority of v .

To get efficient algorithms in the models of computation we want to consider, we do not know how to efficiently compute clusters as defined above, but will compute the following *restricted clusters up to distance R* instead. For any node v of priority i , let

$$C(v, \mathcal{A}, R, G) = \{u \in V \mid d(u, v, G) < d(u, A_{i+1}, G) \text{ and } d(u, v, G) \leq R\}.$$

3.1.1 Computing Priorities \mathcal{A}

The performance of our algorithms relies on the total size of the clusters, which in turn depends on how we compute nodes' priorities. If randomization is allowed, we can use the following algorithm due to Thorup and Zwick [TZ05, TZ06]: set $A_0 = V$ and $A_p = \emptyset$, and for $1 \leq i \leq p-1$ obtain A_i by picking each node from A_{i-1} with probability $(\ln n/n)^{1/p}$. It can be argued that for $\mathcal{A} = (A_i)_{0 \leq i \leq p}$ the size of all clusters, i.e., $\sum_{v \in V} |C(v, \mathcal{A}, G)|$, is $O(pn^{1+1/p})$ in expectation. We now explain how to deterministically compute the priorities of nodes (given by a hierarchy of sets of nodes $\mathcal{A} = (A_i)_{0 \leq i \leq p}$) such that the total size of the resulting clusters is $O(pn^{1+1/p})$.

Thorup and Zwick [TZ05] introduced the notion of *bunches* to analyze the sizes of clusters. For every node $u \in V$, we define the bunch and, for every $0 \leq i \leq p-1$, the i -bunch, both restricted to distance R , as follows:

$$\begin{aligned} B_i(u, \mathcal{A}, R, G) &= \{v \in A_i \setminus A_{i+1} \mid d(u, v, G) < d(u, A_{i+1}, G) \text{ and } d(u, v, G) \leq R\} \\ B(u, \mathcal{A}, R, G) &= \bigcup_{0 \leq i \leq p-1} B_i(u, \mathcal{A}, R, G) \end{aligned}$$

The crucial insight is that $v \in B(u, \mathcal{A}, R, G)$ if and only if $u \in C(v, \mathcal{A}, R, G)$. Thus, it suffices to choose a hierarchy of sets A_i such that $|B_i(u, \mathcal{A}, R, G)| \leq O(n^{1/p})$ for every $u \in V$ and $0 \leq i \leq p-1$.

Our algorithm for deterministically computing this hierarchy of sets of nodes follows the main idea of Roditty, Thorup, and Zwick [RTZ05]. Its pseudocode is given in Procedure 1. As a subroutine this algorithm solves a weighted source detection problem, i.e., for suitable parameters q , A , and R , it computes for every node v the set $L(v, A, R, q, G)$ containing the q nodes of A that are closest to v – up to distance R ; if there are fewer than q nodes of A in distance R to v , $L(v, A, R, q, G)$ contains all of them. Our algorithm for constructing the hierarchy of sets $(A_i)_{0 \leq i \leq p}$ is as follows. We set $A_0 = V$ and $A_p = \emptyset$ and to construct the set A_{i+1} given the set A_i for $0 \leq i \leq p-2$ we first find for each node $v \in V$ the set $L(v, A_i, R, q, G)$ using a source detection algorithm. Then we view the collection of sets $\{L(v, A_i, R, q, G)\}_{v \in V}$ as an instance of the hitting set problem over the universe A_i , where

we want to find a set $A_{i+1} \subseteq A_i$ of minimal size such that each set $L(v, A_i, R, q, G)$ contains at least one node of A_{i+1} . We let A_{i+1} be a hitting set of approximately minimum size⁹ determined by the deterministic greedy heuristic (always adding the element “hitting” the largest number of “un-hit” sets) to produce a hitting set within a factor of $1 + \ln n$ of the optimum [Joh74, ADP80]. In the following we prove the desired bound on the size of the bunches, which essentially requires us to argue that setting $q = \tilde{O}(n^{1/p})$ is sufficient.

Procedure 1: PRIORITIES(G, p, R)

Input: Weighted graph $G = (V, E)$ with positive integer edge weights, number of priorities $p \geq 2$, distance range $R \geq 1$

Output: Hierarchy of sets $(A_i)_{0 \leq i \leq p}$

```

1  $q \leftarrow \lceil 2n^{1/p} \ln(3n)(1 + \ln n) \rceil$ 
2  $A_0 \leftarrow V$ 
3 for  $i = 0$  to  $p - 2$  do
4   Compute  $L(v, A_i, R, q, G)$  for every node  $v \in V$  using a source detection algorithm
5    $\mathcal{C} \leftarrow \emptyset$ 
6   foreach  $v \in V$  do
7     if  $|L(v, A_i, q, R, G)| = q$  then  $\mathcal{C} \leftarrow \mathcal{C} \cup \{L(v, A_i, R, q, G)\}$ 
8   Compute an approximately minimum hitting set  $A_{i+1} \subseteq A_i$  using a greedy heuristic
9  $A_p \leftarrow \emptyset$ 
10 return  $\mathcal{A} = (A_i)_{0 \leq i \leq p}$ 

```

Lemma 3.1 (Implicit in [RTZ05]). *Given a finite collection of sets $\mathcal{C} = \{S_1, \dots, S_q\}$ over a universe U and a parameter $x \geq 1$ such that $|S_i| \geq 2x \ln 3q$ for all $1 \leq i \leq q$, there exists a hitting set $T \subseteq U$ of size $|T| \leq |U|/x$ such that $T \cap S_i \neq \emptyset$ for all $1 \leq i \leq q$.*

We give a proof of Lemma 3.1 in Appendix B for completeness.

Lemma 3.2. *Procedure 1 computes a hierarchy $\mathcal{A} = (A_i)_{0 \leq i \leq p}$ of sets of nodes such that*

$$\sum_{u \in V} C(u, \mathcal{A}, R, G) = \sum_{u \in V} B(u, \mathcal{A}, R, G) = O(pn^{1+1/p}).$$

Proof. We first show by induction that $|A_i| \leq n^{1-i/p}$ for all $0 \leq i \leq p-1$. If $i = 0$ the claim is trivially true because we set $A_0 = V$. We now assume that $|A_i| \leq n^{1-i/p}$ and argue that $|A_i| \leq n^{1-(i+1)/p}$. Our algorithm approximately computes a minimum hitting set of the collection of sets \mathcal{C} containing each set $L(v, A_i, q, R, G)$ of size q . By Lemma 3.1 we know that there is a hitting set A' for \mathcal{C} of size at most $|A_i|/(n^{1/p}(1 + \ln n)) \leq n^{1-i/p}/(1 + \ln n)$ and thus the minimum hitting set A_{i+1} computed by the greedy heuristic has size at most $(1 + \ln n)|A'| \leq n^{1-i/p}$. Note that each set $L(v, A_i, q, R, G)$ might have been empty and in this case the algorithm would have computed $A_{i+1} = \emptyset$, the trivial hitting set.

⁹In principle, local computation is free in the models considered in this paper and we could thus compute a minimum hitting set exactly. However, we decided to present the algorithm in a way that avoids solving NP-complete problems by local computation.

We now show that for every node $u \in V$ and every $0 \leq i \leq p-1$, $B_i(u, \mathcal{A}, R, G) \leq q = O(n^{1/p} \log n)$, which immediately implies the desired bound on the total size of the bunches and clusters. We argue by a simple case distinction that $B_i(u, \mathcal{A}, R, G) \subseteq L(u, A_i, R, q, G)$ and thus, by the definition of the set of the q closest nodes in A_i , $|B_i(u, \mathcal{A}, R, G)| \leq |L(u, A_i, R, q, G)| \leq q$. If $|L(u, A_i, R, q, G)| < q$, then clearly $L(u, A_i, R, q, G) = \{v \in A_i \mid d(u, v, G) \leq R\} \supseteq B_i(u, \mathcal{A}, R, G)$. Otherwise we have $|L(u, A_i, R, q, G)| = q$ and, as the algorithm computed a suitable hitting set, we have $B_i(u, \mathcal{A}, R, G) \subseteq L(u, A_i, R, q, G)$. \square

As an alternative to the algorithm proposed above, the hitting sets can also be computed with the deterministic algorithm of Roditty, Thorup, and, Zwick [RTZ05] which produces so-called “early hitting sets”. For this algorithm we have to set $q = O(n^{1/p} \log n)$, and obtain slightly smaller clusters of total size $O(pn^{1+1/p})$. However, since the logarithmic factors are negligible for our purpose, we have decided to present the simpler algorithm above.

3.1.2 Computing Clusters

Given the priorities of the nodes, the clusters can be computed by finding a shortest-path tree that is “pruned” at nodes whose distance to the root is more than (or equal to) their distance to nodes of higher priority than the root. In the pseudocode of Procedure 2 we formulate this algorithm as a variant of weighted breadth-first search. We will not analyze the performance of this algorithm at this point since it depends on the models of computation that simulate it (see Section 3.1.1 and Section 5 for implementations of the algorithm).

We summarize our guarantees with the following theorem.

Theorem 3.3. *Given a weighted graph G with positive integer edge weights and parameters p and R , Procedure 2 computes a hierarchy of sets $\mathcal{A} = (A_i)_{0 \leq i \leq p}$, where $V = A_0 \subseteq A_1 \subseteq \dots \subseteq A_p = \emptyset$, such that $\sum_{v \in V} |C(v, \mathcal{A}, R, G)| = \tilde{O}(pn^{1+1/p})$. It also computes for every node v the set $C(v, \mathcal{A}, R, G)$ and for each node $w \in C(v, \mathcal{A}, R, G)$ the value of $d(v, w, G)$.*

3.2 Hop Reduction with Additive Error

Consider the following algorithm for computing a set of edges F . First, deterministically compute clusters with $p = \lfloor \sqrt{\log n / \log(9/\epsilon)} \rfloor$ priorities (determined by a hierarchy of sets $\mathcal{A} = (A_i)_{0 \leq i \leq p}$) up to distance $R = n^{1/p} \Delta$. Let F be the set containing an edge for every pair $(u, v) \in V^2$ such that $v \in C(u, \mathcal{A}, R, G)$ and set the weight of such an edge $(u, v) \in F$ to $w(u, v, F) = d(u, v, G)$, where the distance is returned by the algorithm for computing the clusters. Procedure 3 gives the pseudocode of this algorithm.

Lemma 3.4. *Let $F \subseteq V^2$ be the set of edges computed by Procedure 3 for a weighted graph $G = (V, E)$ and parameters $\Delta \geq 1$ and $0 < \epsilon \leq 1$. Then F has size $\tilde{O}(pn^{1+1/p})$, where $p = \lfloor \sqrt{(\log n) / (\log(9/\epsilon))} \rfloor$, and in the graph $H = G \cup F$, for every pair of nodes u and v , we have*

$$d^{(p+1)\lceil d(u,v,G)/\Delta \rceil}(u, v, G) \leq (1 + \epsilon)d(u, v, G) + \epsilon n^{1/p} \Delta / (p + 2),$$

i.e., there is a path π' in H of weight $w(\pi', H) \leq (1 + \epsilon)d(u, v, G) + \epsilon n^{1/p} \Delta / (p + 2)$ consisting of $|\pi'| \leq (p + 1)\lceil d(u, v, G) / \Delta \rceil$ edges.

Procedure 2: CLUSTERS(G, p, R)

Input: Weighted graph $G = (V, E)$ with positive integer edge weights, number of priorities $p \geq 2$, distance range $R \geq 1$

Output: Clusters of G as specified in Theorem 3.3

```
1  $(A_i)_{0 \leq i \leq p} \leftarrow \text{PRIORITIES}(G, p, R)$ 
2 For each  $1 \leq i \leq p - 1$  and every node  $v \in V$  compute  $d(v, A_i, R, G)$ 
3 foreach  $u \in V$  do                                     // Compute cluster of every node
    // Initialization
4   Let  $i$  be the priority of  $u$ , i.e.,  $u \in A_i \setminus A_{i+1}$ 
5   foreach  $v \in V$  do  $\delta(u, v) \leftarrow \infty$ 
6    $\delta(u, u) \leftarrow 0$ 
7    $C(u) \leftarrow \emptyset$ 
   // Iteratively add nodes to cluster
8   for  $L = 0$  to  $R$  do
9     foreach node  $v$  with  $\delta(u, v) = L$  do
10      // Check if  $v$  joins cluster of  $u$  at current level
11      if  $\delta(u, v) < d(v, A_{i+1}, R, G)$  then
12         $C(u) \leftarrow C(u) \cup \{v\}$ 
13        foreach  $(v, w) \in E$  do                               // Update neighbors of  $v$ 
14           $\delta'(u, w) \leftarrow (w(v, w, G) + \delta(u, v))$ 
          if  $\delta'(u, w) < \delta(u, w)$  then  $\delta(u, w) \leftarrow \delta'(u, w)$ 
15 return  $(C(v), \delta(v, \cdot))_{v \in V}$ 
```

Procedure 3: HOPREDUCTIONADDITIVEERROR(G, Δ, ϵ)

Input: Graph $G = (V, E)$ with non-negative integer edge weights, $\Delta \geq 1$, $0 < \epsilon \leq 1$

Output: Hop-reducing set of edges $F \subseteq V^2$ as specified in Lemma 3.4

```
1  $p \leftarrow \left\lfloor \sqrt{\frac{\log n}{\log(9/\epsilon)}} \right\rfloor$ 
2  $R \leftarrow n^{1/p} \Delta$ 
3  $F \leftarrow \emptyset$ 
4  $(C(v), \delta(v, \cdot))_{v \in V} \leftarrow \text{CLUSTERS}(G, p, R)$ 
5 foreach  $u \in V$  do
6   foreach  $v \in C(u)$  do
7      $F \leftarrow F \cup \{(u, v)\}$ 
8      $w(u, v, F) = \delta(u, v)$ 
9 return  $F$ 
```

We devote the rest of this section to proving Lemma 3.4. The bound on the size of F immediately follows from Theorem 3.3. We analyze the hop-reducing properties of F by showing the following. Let π be a shortest path from u to v in G . Then there is a node w on π and a path π' from u to w in $H = G \cup F$ with the following properties:

- (1) The distance from u to w in G is at least Δ .
- (2) The path π' consists of at most p edges of F and at most one edge of G .
- (3) The ratio between the weight of π' in H and the distance from u to w in G is at most $(1 + \epsilon)$ if $w \neq v$ and if $w = v$ then the weight of π' in H is at most β (for some β that we set later).

When we go from u to w using the path π' instead of the subpath of π we are using a “shortcut” of at most $p + 1$ hops that brings us closer to v by a distance of at least Δ at the cost of some approximation. Conditions (1) and (2) guarantee that by repeatedly applying this shortcutting we can find a path π'' from u to v that has at most $(p + 1)\lceil d(u, v, G)/\Delta \rceil$ hops (as we replace subpaths of π with weight at least Δ by paths with at most $p + 1$ hops). Condition (3) guarantees that the multiplicative error introduced by using the shortcut is at most $1 + \epsilon$, except possibly for the last time such a shortcut is used, where we allow an additive error of β . We will show that we can guarantee a value of β that is bounded by $\epsilon n^{1/p}/(p + 2)$. This type of analysis has been used before by Thorup and Zwick [TZ06] to obtain a spanner for unweighted graphs defined from the partial shortest path trees of the clusters, but without considering the hop-reduction aspect. Bernstein [Ber09] also used a similar analysis to obtain a hop set for weighted graphs using clusters with full distance range. We previously used this type of analysis to obtain a randomized hop set which is not based on clusters, but on a similar notion [HKN14].

To carry out the analysis as explained above we define a value r_i for every $0 \leq i \leq p - 1$ as follows:

$$\begin{aligned} r_0 &= \Delta \\ r_i &= \frac{(4 + 2\epsilon) \sum_{0 \leq j \leq i-1} r_j}{\epsilon}. \end{aligned}$$

The intuition is that a node u of priority i tries to take an edge of F to shortcut the way to v by at least r_i . If this fails it will find an edge in F going to a node v' of higher priority. Thus, to fulfill Condition (3), v' has to try and shortcut even more “aggressively”. Consequently, the values of r_i grow exponentially with the priority i .

We have chosen the range of the clusters large enough such that nodes of the highest priority will always find the desired shortcut edge in F . We will show that the additive error incurred by this strategy is at most

$$\beta = \sum_{0 \leq i \leq p-1} 2r_i.$$

This value can in turn be bounded as follows.

Lemma 3.5. $\beta \leq \epsilon n^{1/p} \Delta / (p + 2)$.

Proof. We first show that, for all $0 \leq i \leq p-1$, $\sum_{0 \leq j \leq i} r_j \leq 7^i \Delta / \epsilon^i$. The proof is by induction on i . For $i = 0$ we have $r_0 = \Delta = 7^0 \Delta$ by the definition of r_0 and for $i \geq 1$ we use the inequality $\epsilon \leq 1$ and the induction hypothesis as follows:

$$\begin{aligned} \sum_{0 \leq j \leq i} r_j &= \sum_{0 \leq j \leq i-1} r_j + r_i = \sum_{0 \leq j \leq i-1} r_j + \frac{(4 + 2\epsilon) \sum_{0 \leq j \leq i-1} r_j}{\epsilon} \\ &\leq \frac{\sum_{0 \leq j \leq i-1} r_j}{\epsilon} + \frac{6 \sum_{0 \leq j \leq i-1} r_j}{\epsilon} \leq \frac{7 \sum_{0 \leq j \leq i-1} r_j}{\epsilon} \leq \frac{7 \cdot 7^{i-1} \Delta}{\epsilon \cdot \epsilon^{i-1}} \leq \frac{7^i \Delta}{\epsilon^i}. \end{aligned}$$

Using this inequality and the fact that $(2p+7)7^{p-1} \leq 9^p$ for all $p \geq 0$ we get

$$\frac{(p+2)\beta}{\epsilon} = \frac{(p+2) \sum_{0 \leq j \leq p-1} 2r_j}{\epsilon} \leq \frac{(2p+4)7^{p-1} \Delta}{\epsilon^p} \leq \frac{9^p \Delta}{\epsilon^p} \leq n^{1/p} \Delta$$

The last inequality holds as by our choice of $p = \lfloor \sqrt{\log n / \log(9/\epsilon)} \rfloor$:

$$(9/\epsilon)^p = 2^{p \cdot \log(9/\epsilon)} \leq 2^{\frac{\sqrt{\log n}}{\sqrt{\log(9/\epsilon)}} \cdot \log(9/\epsilon)} = 2^{\sqrt{\log n} \cdot \sqrt{\log(9/\epsilon)}} = 2^{\frac{\sqrt{\log(9/\epsilon)}}{\sqrt{\log n}} \cdot \log n} = 2^{\log n \cdot (1/p)} = n^{1/p}. \quad \square$$

In the following we fix some values of ϵ and Δ and let F denote the set of edges computed by Procedure 3. We now show that F has a certain structural property before we carry out the hop-reduction proof.

Lemma 3.6. *Let u and v be nodes such that $u \in A_i \setminus A_{i+1}$ has priority i and $d(u, v, G) \leq r_i$. Either*

- (1) F contains an edge (u, v) of weight $w(u, v, F) = d(u, v, G)$ or
- (2) F contains an edge (u, v') to a node $v' \in A_{i+1}$ of priority $j \geq i+1$ of weight $w(u, v', F) \leq 2r_i$.

Proof. Consider first the case $v \in C(u, \mathcal{A}, R, G)$. Then F contains the edge (u, v) of weight $w(u, v, F) = d(u, v, G)$.

Consider now the case $v \notin C(u, \mathcal{A}, R, G)$. Note that by the definition of β we have $r_i \leq \beta/2 < \beta$ and by Lemma 3.5 we have $\beta \leq n^{1/p} \Delta$. As the algorithm sets $R = n^{1/p} \Delta$ we have $r_i \leq R$ by Lemma 3.5 and thus $d(u, v, G) \leq R$. From the definition of $C(u, \mathcal{A}, R, G)$ it now follows that $d(v, u, G) \geq d(v, A_{i+1}, G)$. Thus there exists some node $v_1 \in A_{i+1}$ of priority $p_1 \geq i+1$ such that $d(v, v_1, G) \leq d(u, v, G)$. By the triangle inequality we get $d(u, v_1, G) \leq d(u, v, G) + d(v, v_1, G) \leq 2d(u, v, G) \leq 2r_i \leq R$. If $u \in C(v_1, \mathcal{A}, R, G)$ then we are done as F contains the edge (u, v_1) of weight $w(u, v_1, F) = d(u, v_1, G) \leq 2r_i$. Otherwise it follows from the definition of $C(v_1, \mathcal{A}, R, G)$ that there is some node $v_2 \in A_{p_1+1}$ of priority $p_2 \geq p_1+1 \geq i+1$ such that $d(u, v_2, G) \leq d(u, v_1, G) \leq 2r_i \leq R$. By repeating the argument above we want to find some node $v_j \in A_{i+1}$ of priority $p_j \geq i+1$ such that $d(u, v_j, G) \leq d(u, v, G) \leq 2r_i$. As for every node $v' \in A_{p-1}$ of priority $p-1$, $C(v', \mathcal{A}, R, G)$ contains all nodes that are at distance at most R from v' in G this repeated argument stops eventually and we find such a node. \square

To finish the proof of Lemma 3.4 we show in the next lemma that F has the properties we demanded, i.e., in the shortcut graph H which consists of G and the additional edges of F , we can approximate shortest paths using a reduced number of hops.

Lemma 3.7. *For every pair of nodes $u, v \in V$ such that $d(u, v, G) < \infty$, the graph $H = G \cup F$ contains a path π' from u to v of weight $w(\pi', H) \leq (1 + \epsilon)d(u, v, G) + \beta$ consisting of $|\pi'| \leq (p + 1)\lceil d(u, v, G)/\Delta \rceil$ edges.*

Proof. The proof is by induction on the distance from u to v in G . The claim is trivially true for the base case $d(u, v, G) = 0$ in which $u = v$. Thus, we only need to consider the induction step in which $d(u, v, G) \geq 1$.

Let π denote the shortest path from u to v in G . We now define a sequence of nodes u_0, u_1, \dots, u_l . For every $0 \leq j \leq l$, we denote by p_j the priority of u_j . We set $u_0 = u$ and, given u_j , we define u_{j+1} as follows. Let w be the node on π closest to v that is at distance at most r_{p_j} from u_j in G (this node might be v itself). If H contains the edge (u_j, w) we stop (and set $l = j$). Otherwise we know by Lemma 3.6 that H contains an edge (u_j, u') to a node u' of priority at least $p_j + 1$. In that case we set $u_{j+1} = u'$. We know further by Lemma 3.6 that $d(u_j, u_{j+1}, G) \leq 2r_{p_{j+1}-1}$. Having defined the sequence u_0, u_1, \dots, u_l , we denote by w the node on π closest to v that is at distance at most r_{p_l} from u_l in G (again, this node might be v itself). Figure 1 illustrates the definition of this sequence.

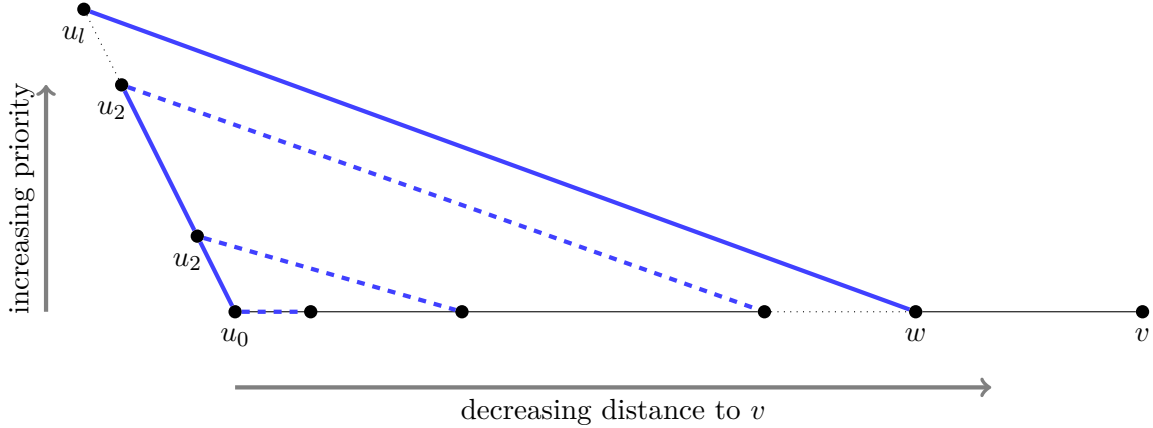


Figure 1: Schematic illustration of the definition of the sequence of nodes u_0, u_1, \dots, u_l, w . The bottom line represents the shortest path from u to v . The thick, blue edges are the edges of F used to shorten the distance to v . The dashed, blue edges are not contained in F and imply the existence of edges to nodes of increasing priority. The dotted lines indicate repetitions that are omitted in the picture.

Consider first the case that $w = v$. Let π' denote the path $\langle u_0, \dots, u_l, w \rangle$. This path has at most p hops and since $d(u, v, G) \geq 1$ we trivially have $p \leq (p + 1)\lceil d(u, v, G)/\Delta \rceil$. Furthermore we can bound the weight of π' as follows:

$$\begin{aligned} w(\pi', H) &= \sum_{0 \leq j \leq l-1} w(u_j, u_{j+1}, H) + w(u_l, w, H) \leq \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) + d(u_l, w, G) \\ &\leq \sum_{0 \leq j \leq l-1} 2r_{p_{j+1}-1} + r_{p_l} \end{aligned}$$

$$\begin{aligned}
&\leq \sum_{0 \leq j \leq p-1} 2r_j \\
&= \beta \leq (1 + \epsilon)d(u, v, G) + \beta.
\end{aligned}$$

Consider now the case $w \neq v$. Let w' be the neighbor of w on π (that in G is closer to v than w is). We will define the path π' from u to v as the concatenation of two paths π_1 and π_2 . Let π_1 be the path $\langle u_0, \dots, u_l, w, w' \rangle$. We will define the path π_2 from w' to v later on. Note that π_1 consists of $|p_1| \leq p + 1$ hops. We will now show that

$$w(\pi_1, H) \leq (1 + \epsilon)d(u, w', G). \quad (5)$$

In order to get this bound we will need some auxiliary inequalities. By Lemma 3.6 we have, for all $0 \leq j \leq l - 1$,

$$d(u_j, u_{j+1}, G) \leq 2r_{p_{j+1}-1} \quad (6)$$

and by the definition of r_{p_l} we have

$$\epsilon r_{p_l} = (4 + 2\epsilon) \sum_{0 \leq j \leq p_l-1} r_j. \quad (7)$$

Remember that w is the node on π closest to v that is at distance at most r_{p_j} from u_l in G . Since the neighbor w' of w is closer to v than w is, this definition of w guarantees that $d(u_l, w', G) > r_{p_j}$. As $d(u_l, w', G) \leq d(u_l, w, G) + d(w, w', G)$ by the triangle inequality, we have

$$d(u_l, w, G) + d(w, w', G) > r_{p_j}. \quad (8)$$

By the triangle inequality we also have

$$d(u_l, w, G) \leq \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) + d(u, w, G).$$

and thus

$$d(u_l, w, G) - \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) \leq d(u, w, G). \quad (9)$$

We now obtain Inequality (5) as follows:

$$\begin{aligned}
w(\pi_1, H) &= \sum_{0 \leq j \leq l-1} w(u_j, u_{j+1}, H) + w(u_l, w, H) + w(w, w', H) \\
&= \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) + d(u_l, w, G) + d(w, w', G) \\
&= (2 + \epsilon) \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) + d(u_l, w, G) + d(w, w', G) - (1 + \epsilon) \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) \\
&\stackrel{(6)}{\leq} (2 + \epsilon) \sum_{0 \leq j \leq l-1} 2r_{p_{j+1}-1} + d(u_l, w, G) + d(w, w', G) - (1 + \epsilon) \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) \\
&\leq (2 + \epsilon) \sum_{0 \leq j \leq p_l-1} 2r_j + d(u_l, w, G) + d(w, w', G) - (1 + \epsilon) \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) \\
&\stackrel{(7)}{=} \epsilon r_{p_l} + d(u_l, w, G) + d(w, w', G) - (1 + \epsilon) \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G)
\end{aligned}$$

$$\begin{aligned}
& \stackrel{(8)}{<} \epsilon(d(u_l, w, G) + d(w, w', G)) + d(u_l, w, G) + d(w, w', G) - (1 + \epsilon) \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) \\
&= (1 + \epsilon) \left(d(u_l, w, G) - \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) \right) + (1 + \epsilon)d(w, w', G) \\
& \stackrel{(9)}{\leq} (1 + \epsilon)d(u, w, G) + (1 + \epsilon)d(w, w', G) \\
&\leq (1 + \epsilon)(d(u, w, G) + d(w, w', G)) = (1 + \epsilon)d(u, w', G)
\end{aligned}$$

Note that $d(w', v, G) < d(u, v, G)$. Therefore we may apply the induction hypothesis on w' and get that the graph H contains a path π_2 of weight $w(\pi_2, H) \leq (1 + \epsilon)d(w', v, G) + \beta$ that has $|\pi_2| \leq (p + 1)\lceil d(w', v, G)/\Delta \rceil$ hops. Let π' denote the concatenation of π_1 and π_2 . Then π' is a path from u to v in H of weight

$$\begin{aligned}
w(\pi', H) &= w(\pi_1, H) + w(\pi_2, H) \\
&\leq (1 + \epsilon)d(u, w', G) + (1 + \epsilon)d(w', v, G) + \beta \\
&= (1 + \epsilon)(d(u, w', G) + d(w', v, G)) + \beta \\
&= (1 + \epsilon)d(u, v, G) + \beta.
\end{aligned}$$

It remains to bound the number of hops of π' . To get the desired bound we first show that $d(u, w', G) \geq \Delta$. By the triangle inequality we have

$$d(u_l, w', G) \leq \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) + d(u, w', G).$$

As argued above, we have $d(u_l, w', G) > r_{p_j}$ and $\sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) \leq \sum_{0 \leq j \leq p_l-1} 2r_j$. By the definition of r_{p_l} we therefore get:

$$\begin{aligned}
d(u, w', G) &\geq d(u_l, w', G) - \sum_{0 \leq j \leq l-1} d(u_j, u_{j+1}, G) \\
&\geq r_{p_l} - \sum_{0 \leq j \leq p_l-1} 2r_j = (4/\epsilon) \sum_{0 \leq j \leq p_l-1} r_j \geq r_0 = \Delta.
\end{aligned}$$

Now that we know that $d(u, w', G) \geq \Delta$, or equivalently $d(u, w', G)/\Delta \geq 1$, we get the following for counting the number of hops of π' by adding the number of hops of π_1 to the number of hops of π_2 :

$$\begin{aligned}
|\pi'| &= |\pi_1| + |\pi_2| \leq p + 1 + (p + 1)\lceil d(w', v, G)/\Delta \rceil = (p + 1)(1 + \lceil d(w', v, G)/\Delta \rceil) \\
&= (p + 1)\lceil 1 + d(w', v, G)/\Delta \rceil \\
&\leq (p + 1)\lceil d(u, w', G)/\Delta + d(w', v, G)/\Delta \rceil \\
&= (p + 1)\lceil (d(u, w', G) + d(w', v, G))/\Delta \rceil \\
&= (p + 1)\lceil d(u, v, G)/\Delta \rceil
\end{aligned}$$

Thus, π' has the desired number of edges. □

3.3 Hop Reduction without Additive Error

Consider a shortest path π from u to v with h hops and weight $R \geq \Delta$. With the hop reduction of Procedure 3 we can compute a set of edges F such that in $G \cup F$ we find a path from u to v with $\tilde{O}(R/\Delta)$ hops of weight approximately R . We now use the weight-rounding technique of Lemma 2.2 and repeatedly apply this algorithm to obtain a set of edges F such that in $G \cup F$ there is a path from u to v with $O(h/\Delta)$ hops and weight approximately R . As in general R can only be upper-bounded by nW (where W is the maximum edge weight of G) and h can be upper-bounded by n , it is clear that the second type of hop reduction is more desirable. Additionally, if h is sufficiently larger than Δ , then the additive error inherent in the hop reduction of Procedure 3 can be counted as an additional multiplicative error of ϵ .

The second hop reduction algorithm roughly works as follows. For every possible distance range of the form $2^j \dots 2^{j+1}$ we scale down the edge weights of G by a certain factor and run the algorithm of Procedure 3 on the modified graph \hat{G}_j to compute a set of edges \hat{F}_j . We then simply return the union of all these edge sets (with the weights scaled back to normal again). Procedure 4 shows the pseudocode of this algorithm.

Procedure 4: HOPREDUCTION($G, \Delta, h, \epsilon, W$)

Input: Weighted graph $G = (V, E)$ with integer edge weights from 1 to W , $\Delta \geq 1$, $h \geq 1$, $0 < \epsilon \leq 1$

Output: Hop-reducing set of edges $F \subseteq V^2$ as specified in Lemma 3.8

```

1  $\epsilon' \leftarrow \epsilon/6$ 
2  $\Delta' \leftarrow 3\Delta/\epsilon'$ 
3  $F \leftarrow \emptyset$ 
4 for  $j = 0$  to  $\lfloor \log(nW) \rfloor$  do
5    $\hat{G}_j \leftarrow (V, E)$ 
6    $\rho_j \leftarrow \epsilon' 2^j / h$ 
7   foreach  $(u, v) \in E$  do  $w(u, v, \hat{G}_j) \leftarrow \lceil w(u, v, E) / \rho_j \rceil$ 
8    $\hat{F}_j \leftarrow \text{HOPREDUCTIONADDITIVEERROR}(\hat{G}_j, \Delta', \epsilon')$ 
9   foreach  $(u, v) \in \hat{F}_j$  do
10     $F \leftarrow F \cup \{(u, v)\}$ 
11     $w(u, v, F) \leftarrow \min(w(u, v, \hat{F}_j) \cdot \rho_j, w(u, v, G))$ 
12 return  $F$ 
```

Lemma 3.8. Let $F \subseteq V^2$ be the set of edges computed by Procedure 4 for a weighted graph $G = (V, E)$ and parameters $\Delta \geq 1$, $h \geq 1$, and $0 < \epsilon \leq 1$. Then F has size $\tilde{O}(pn^{1+1/p} \log nW)$, where $p = \lfloor \sqrt{(\log n)/(\log(54/\epsilon))} \rfloor$, and if $h \geq n^{1/p} \Delta / (p+2)$, then in the graph $H = G \cup F$ we have, for every pair of nodes u and v ,

$$d^{(p+2)h/\Delta}(u, v, H) \leq (1 + \epsilon) d^h(u, v, G).$$

Proof. Let u and v be a pair of nodes and set $j = \lfloor \log d^h(u, v, G) \rfloor$, i.e., $2^j \leq d^h(u, v, G) \leq 2^{j+1}$. Let π be a shortest $\leq h$ hop path in G , i.e., π has weight $w(\pi, G) = d^h(u, v, G)$ and

π consists of $|\pi| \leq h$ hops. The algorithm sets $\epsilon' = \epsilon/6$ and uses a graph \widehat{G}_j which has the same nodes and edges as G , but in which every edge weight is first scaled down by a factor of $\rho_j = \epsilon'2^j/h$ and then rounded up to the next integer. By Lemma 2.2 we have $d(u, v, \widehat{G}_j) \cdot \rho_j \leq (1 + \epsilon')d^h(u, v, G)$ and $d(u, v, \widehat{G}_j) \leq (1 + 2/\epsilon')h \leq 3h/\epsilon'$.

Since $d^h(u, v, G) \leq nW$ the algorithm has computed a set of edges \widehat{F}_j . By Lemma 3.4, there is a path π' in $\widehat{H}_j = \widehat{G}_j \cup \widehat{F}_j$ of weight at most $(1 + \epsilon')d(u, v, \widehat{G}_j) + \epsilon'n^{1/p}\Delta'/(p+2)$ and with at most $|\pi'| \leq (p+1)\lceil d(u, v, \widehat{G}_j)/\Delta' \rceil$ hops. Since $d(u, v, \widehat{G}_j) \leq 3h/\epsilon'$ and the algorithm sets $\Delta' = 3\Delta/\epsilon'$ we have

$$\begin{aligned} |\pi'| &\leq (p+1) \cdot \left\lceil \frac{d(u, v, \widehat{G}_j)}{\Delta'} \right\rceil \leq (p+1) \cdot \left\lceil \frac{3h}{\epsilon'\Delta'} \right\rceil = (p+1) \cdot \left\lceil \frac{h}{\Delta} \right\rceil \leq (p+1) \left(\frac{h}{\Delta} + 1 \right) \\ &= \frac{(p+1)h}{\Delta} + (p+1) \leq \frac{(p+1)h}{\Delta} + 4^p \leq \frac{(p+1)h}{\Delta} + n^{1/p} \leq \frac{(p+1)h}{\Delta} + \frac{h}{\Delta} = \frac{(p+2)h}{\Delta}. \end{aligned}$$

The algorithm “scales back” the edge weights of \widehat{F}_j when adding them to F and thus $w(u, v, F) \leq w(u, v, \widehat{F}_j) \cdot \rho_j$. We now argue that $d^{(p+1)\lceil h/\Delta \rceil}(u, v, H) \leq (1 + \epsilon')d^h(u, v, G)$ by bounding the weight of π' in $H = G \cup F$. For every edge (u, v) of π' we have $w(u, v, H) \leq w(u, v, F) \leq w(u, v, \widehat{F}_j) \cdot \rho_j$ if $(u, v) \in \widehat{F}_j$ and $w(u, v, H) \leq w(u, v, G) \leq w(u, v, \widehat{G}_j) \cdot \rho_j$ otherwise. Thus, $w(\pi', H) \leq w(\pi', \widehat{H}_j) \cdot \rho_j$ and together with the assumption $h \geq n^{1/p}\Delta/(p+2)$ we get

$$\begin{aligned} d^{(p+2)h/\Delta}(u, v, H) &\leq w(\pi', H) \leq w(\pi', \widehat{H}_j) \cdot \rho_j \leq \left((1 + \epsilon')d(u, v, \widehat{G}_j) + \frac{\epsilon'n^{1/p}\Delta'}{p+2} \right) \cdot \rho_j \\ &= (1 + \epsilon')d(u, v, \widehat{G}_j) \cdot \rho_j + \frac{\epsilon'n^{1/p}\Delta'\rho_j}{p+2} \\ &= (1 + \epsilon')d(u, v, \widehat{G}_j) \cdot \rho_j + \frac{3\epsilon'2^j n^{1/p}\Delta}{h(p+2)} \\ &\leq (1 + \epsilon')d(u, v, \widehat{G}_j) \cdot \rho_j + 3\epsilon'2^j \\ &\leq (1 + \epsilon')^2 d^h(u, v, G) + 3\epsilon' d^h(u, v, G) \\ &\leq (1 + 6\epsilon') d^h(u, v, G) \\ &= (1 + \epsilon) d^h(u, v, G). \end{aligned} \quad \square$$

3.4 Computing the Hop Set

We finally explain how to repeatedly use the hop reduction of Procedure 4 to obtain an $(n^{o(1)}, o(1))$ -hop set. Procedure 4 computes a set of edges F that reduces the number of hops needed to approximate the distance between any pair of nodes by a factor of $1/\Delta$ (where Δ is a parameter). Intuitively we would now like to use a large value of Δ to compute a hop set. However, we want to avoid large values of Δ for two reasons. The first reason is that F only reduces the number of hops if the shortest path has $h \geq \Delta n^{o(1)}$ hops. Thus, for shortest paths that already have $h < \Delta$ hops the hop reduction is not effective. The second reason is efficiency. The algorithm requires us to compute clusters for distances up to $\Delta n^{o(1)}$ and, in the models of computation we consider later on, we do not know how to do this fast enough for our purposes.

We therefore use the following iterative approach in which we repeatedly apply the hop reduction of Procedure 4 with $\Delta = n^{o(1)}$. We first compute a set of edges F_1 that reduces the number of hops in G by a factor of $1/\Delta$. We then add all these edges to G and consider the graph $H_1 = G \cup F_1$. We apply the algorithm again on H_1 to compute a set of edges F_2 that reduces the number of hops in H_1 by a factor of $1/\Delta$. Now observe that the set of edges $F_1 \cup F_2$ reduces the number of hops in G by a factor of $1/\Delta^2$. We show that by repeating this process $p = \Theta(\sqrt{\log n / \log(\sqrt{\log n}/\epsilon)})$ times we can compute a set F that reduces the number of hops to $n^{1/p}$. Procedure 5 shows the pseudocode of this algorithm.

Procedure 5: HOPSET(G, ϵ, W)

Input: Weighted graph $G = (V, E)$ with integer edge weights from 1 to W , $0 < \epsilon \leq 1$

Output: $(n^{1/p}, \epsilon)$ -hop set $F \subseteq V^2$ as specified in Theorem 3.9

```

1  $\epsilon' \leftarrow \frac{\epsilon}{2\sqrt{\log n}}$ 
2  $W' \leftarrow (1 + \epsilon)nW$ 
3  $p \leftarrow \lfloor \sqrt{\frac{\log n}{\log(54/\epsilon')}} \rfloor$ 
4  $\Delta \leftarrow (p + 2)n^{1/p}$ 
5  $F \leftarrow \emptyset$ 
6  $H_0 \leftarrow G$ 
7 for  $i = 0$  to  $p - 1$  do
8    $h_i \leftarrow n^{1-i/p}$ 
9    $F_{i+1} \leftarrow \text{HOPREDUCTION}(H_i, \Delta, h_i, \epsilon', W')$ 
10   $F \leftarrow F \cup F_{i+1}$ 
11   $H_{i+1} \leftarrow H_i \cup F_{i+1}$ 
12 return  $F$ 

```

Theorem 3.9. Let $F \subseteq V^2$ be the set of edges computed by Procedure 5 for a weighted graph $G = (V, E)$ and a parameter $0 < \epsilon \leq 1$. Then F is an $(n^{1/p}, \epsilon)$ -hop set of size $\tilde{O}(p^2 n^{1+1/p} \log nW)$, where $p = \lfloor \sqrt{(\log n)/(\log(108\sqrt{\log n}/\epsilon))} \rfloor$.

Proof. The algorithm sets $\epsilon' = \epsilon/(2\sqrt{\log n})$ and $p = \lfloor \sqrt{(\log n)/(\log(54/\epsilon'))} \rfloor$ and uses a parameter $h_i = n^{1-i/p}$ for each graph H_i . For every $0 \leq i \leq p - 2$ we set $h_i = n^{1-i/p} \geq n^{2/p} = n^{1/p}\Delta/(p + 2)$ and thus, by Lemma 3.8, for every pair of nodes u and v we have

$$d^{h_{i+1}}(u, v, H_{i+1}) = d^{h_i/n^{1/p}}(u, v, H_{i+1}) = d^{(p+2)h_i/\Delta}(u, v, H_{i+1}) \leq (1 + \epsilon')d^{h_i}(u, v, H_i).$$

By iterating this argument we get

$$d^{h_i}(u, v, H_i) \leq (1 + \epsilon')^i d^{h_0}(u, v, H_0) = (1 + \epsilon')^i d^n(u, v, G) = (1 + \epsilon')^i d(u, v, G)$$

for every $1 \leq i \leq p - 1$ and now in particular for $i = p - 1$ we have

$$d^{n^{1/p}}(u, v, H_{p-1}) = d^{h_{p-1}}(u, v, H_{p-1}) \leq (1 + \epsilon')^{(p-1)} d(u, v, G).$$

Finally, since $p - 1 \leq \sqrt{\log n}$ we have, by Lemma 3.10 below,

$$(1 + \epsilon')^{p-1} = \left(1 + \frac{\epsilon}{2\sqrt{\log n}}\right)^{p-1} \leq \left(1 + \frac{\epsilon}{2\sqrt{\log n}}\right)^{\sqrt{\log n}} \leq 1 + \epsilon.$$

As $H_{p-1} = G \cup F$ it follows that $d^{n^{1/p}}(u, v, G \cup F) \leq (1 + \epsilon)d(u, v, G)$ and thus $F = \bigcup_{1 \leq i \leq p-1} F_i$ is an $(n^{1/p}, \epsilon)$ -hop set. \square

Lemma 3.10. *For all $0 \leq x \leq 1$ and all $y > 0$,*

$$\left(1 + \frac{x}{2y}\right)^y \leq 1 + x.$$

Proof. Let e denote Euler's constant. We will use the following well-known inequalities: $(1 + 1/z)^z \leq e$ (for all $z > 0$), $e^z \leq 1/(1 - z)$ (for all $z < 1$), and $1/(1 - z) \leq 1 + 2z$ (for all $0 \leq z \leq 1/2$). We then get:

$$\left(1 + \frac{x}{2y}\right)^y = \left(\left(1 + \frac{x}{2y}\right)^{\frac{2y}{x}}\right)^{\frac{x}{2}} \leq e^{\frac{x}{2}} \leq \frac{1}{1 - \frac{x}{2}} \leq 1 + x. \quad \square$$

The main computational cost for constructing the hop set comes from computing the clusters in Procedure 3, which is used as a subroutine repeatedly. Observe that in total it will perform $O(p \log n W)$ calls to Procedure 2 to compute clusters, each with $p = \Theta(\sqrt{(\log n)/(\log(\sqrt{\log n}/\epsilon))})$ priorities and distance range $R = O(p\sqrt{\log n} n^{2/p}/\epsilon)$ on a weighted graph of size $\tilde{O}(m + p^2 n^{1+1/p} \log(nW))$. Note that if $1/\epsilon \leq \text{polylog } n$, then $n^{1/p} = n^{o(1)}$. Thus, Procedure 5 will then compute an $(n^{o(1)}, o(1))$ -hop set of size $O(n^{1+o(1)} \log W)$ and it will perform $\tilde{O}(\log W)$ cluster computations with $p = \Theta(\sqrt{\log n / \log \log n})$ priorities up to distance range $O(n^{o(1)})$ on graphs of size $O(m^{1+o(1)} \log W)$ each.

4 Distributed Single-Source Shortest Paths Algorithm on Networks with Arbitrary Topology

In this section we describe a deterministic distributed algorithm for computing distances from a source node s . It consists of two parts. The first part is constructing a suitable *overlay network*. A *randomized* construction algorithm was given in [Nan14] such that it was sufficient to solve SSSP on the resulting overlay network in order to solve the same problem on the whole network. We give a *deterministic* version of this result in Section 4.1. The second part is a more efficient algorithm for computing SSSP on an overlay network using Procedures 2 to 5 from before (see Section 4.2). In Section 4.3, we show how to finish the computation following [Nan14].

4.1 Computing an Overlay Network Deterministically

An *overlay network* (also known as *landmark* or *skeleton* [Som14, LPS13]) as defined in [Nan14] is a *virtual* network G' of nodes and “virtual edges” that is built on top of an underlying *real* network G ; i.e., $V(G') \subseteq V(G)$ and $E(G') = V(G') \times V(G')$ such that the

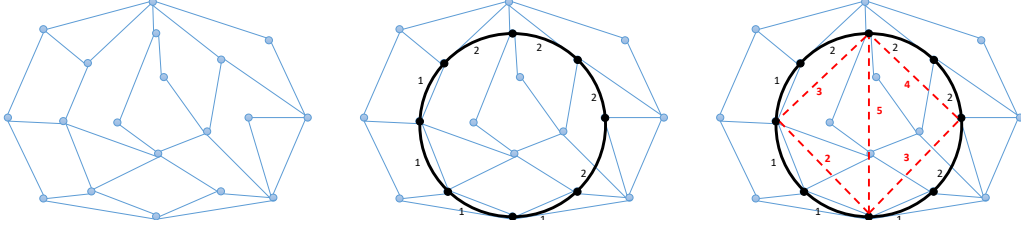


Figure 2: An overview of the main steps of our algorithm. The left picture depicts the input graph. Thick edges and nodes (in black) in the middle picture depicts a possible overlay network. Dashed edges (in red) in the right picture depicts a possible hop set of the overlay network.

weight of an edge in G' is an approximation of the distance of its endpoints in G and is ∞ if no path exists between them in G . The nodes in $V(G')$ are called *centers*. Computing G' means that after the computation every node in G' knows whether it is a center and knows all virtual edges to its neighbors in G' and the corresponding weights. We show in this subsection that there is a $\tilde{O}(\sqrt{n})$ -time algorithm that constructs an overlay network G' of $\tilde{O}(\sqrt{n}/\epsilon)$ nodes such that a $(1 + \epsilon)$ -approximation to SSSP in G' , can be converted to a $(1 + 7\epsilon)$ -approximation to SSSP in G , as stated formally below.

Theorem 4.1. *Given any weighted undirected network G with source node s and any parameter $0 < \epsilon \leq 1$, there is a $\tilde{O}(\sqrt{n})$ -time deterministic distributed algorithm that computes an overlay network G' and some additional information for every node with the following properties.*

- Property 1: $|V(G')| = \tilde{O}(\sqrt{n}/\epsilon)$ and $s \in V(G')$.
- Property 2: For every node $u \in V(G)$, as soon as u receives a $(1 + \epsilon)$ -approximation $\tilde{d}(s, v)$ of $d(s, v, G')$ for all centers $v \in V(G')$, it can infer a $(1 + 7\epsilon)$ -approximation of $d(s, u, G)$ without any additional communication.

Before proving the above theorem, we first recall how it was achieved with a randomized algorithm in [Nan14]¹⁰ (see Theorem 4.2 of the arXiv version¹¹ of [Nan14] for details).

- In the first step of [Nan14], the algorithm selects each node to be a center with probability $\tilde{\Theta}(1/\sqrt{n})$ and also makes s a center. By a standard “hitting set” argument (e.g. [UY91, DFI05]), any shortest path containing \sqrt{n} edges will contain a center with high probability. Also, the number of centers is $\tilde{\Theta}(\sqrt{n})$ with high probability.
- In the second step, the algorithm makes sure that every node v knows the $\tilde{\Theta}(\sqrt{n})$ -hop distances between v and all centers using a *light-weight bounded-hop single-source shortest paths algorithm* from all centers in parallel combined with the *random delay technique* to avoid congestion.

¹⁰We note that [Nan14] proved this theorem for general parameters λ and α but we will only need it for $\lambda = \alpha = \sqrt{n}$.

¹¹<http://arxiv.org/pdf/1403.5171v2.pdf>

We derandomize the first step as follows: In Section 4.1.1 we assign to each node u a *type*, denoted by $t(u)$. (To compute these types, we invoke the *source detection algorithm* of Lenzen and Peleg [LP13], as we will explain in Section 4.1.1.) The special property of nodes' types is that every path π containing \sqrt{n} edges contains a node u of a “desired” type; in particular, $t(u)$ is not too big compared to $w(\pi, G)$ (see Lemma 4.2 for details). This is comparable to the property obtained from the hitting set argument, which would be achieved if we made the special node of *every* path a center. However, this may create too many centers (we want the number of centers to be $\tilde{O}(\sqrt{n}/\epsilon)$). Instead we select some nodes to be centers using the *ruling set* algorithm, as described in Section 4.1.2. After this, we get a small set of centers such that every node u of type $t(u)$ is not far from one of the centers. Thus, while we cannot guarantee that the path π *contains* a center, we can guarantee that it contains a node that is *not far* from a center (see Lemma 4.3 for details). To derandomize the second step, we use the recent algorithm of Lenzen and Patt-Shamir [LP15] for the *Partial Distance Estimation* (PDE) problem together with the above Procedures 2 to 5, as we will explain in Section 4.1.3.

Parameters. The parameters used by our algorithm in the following are $\epsilon' = \epsilon/(18\lambda)$, $h = \lfloor \epsilon'\sqrt{n} \rfloor$, $h' = (1 + 2/\epsilon')h \leq 3\sqrt{n}$, $h^* = 9\lambda\sqrt{n}$, $k = 2h^* + 2\sqrt{n}$, and $k' = (1 + 2/\epsilon')k$. Recall that λ is the number of bits used to represent each ID in the network.

4.1.1 Types of Nodes

For any integer i , we let $\rho_i = \frac{\epsilon' 2^i}{h}$ and let G_i be the graph with the same nodes and edges as G and weight $w(u, v, G_i) = \lceil \frac{w(u, v, G)}{\rho_i} \rceil$ for every edge (u, v) . Note that we have chosen h' such that $d(u, v, G_i) \leq h'$ by Equation (3) of Lemma 2.2. For any node u , let the *ball* of u in G_i be $B(u, G_i, h') = \{v \in V(G_i) \mid d(u, v, G_i) \leq h'\}$. Note that for any index i and nodes u and v , $d(u, v, G_{i+1}) \leq d(u, v, G_i)$; thus, $B(u, G_i, h') \subseteq B(u, G_{i+1}, h')$. Let the *type* $t(u)$ of u be the smallest index i such that $|B(u, G_i, h')| \geq h$. We crucially exploit the following structural property.

Lemma 4.2. *For every path π of G consisting of $|\pi| = \sqrt{n}$ edges there is a node u on π such that $2^{t(u)} \leq 2\epsilon'w(\pi, G)$.*

Proof. Let $l = \lceil |\pi|/h \rceil \geq 1/\epsilon$ and let x and y denote the endpoints of π . Partition π into the path π_x consisting of the $(l-1)h$ edges closest to x and the path π_y consisting of the $|\pi| - (l-1)h$ edges closest to y . Further partition π_x into $l-1$ non-overlapping subpaths of exactly h edges, and expand the path π_y by adding edges of π_x to it until it has h edges. Thus, there are now l paths of exactly h edges each and total weight at most $2w(\pi, G)$. It follows that there exists a subpath π' of π consisting of exactly h edges and weight at most $2w(\pi, G)/l \leq 2\epsilon'w(\pi, G)$. Let u and v be the two endpoints of π' and let i be the index such that $2^i \leq d^h(u, v, G) \leq 2^{i+1}$. By Equation (3) of Lemma 2.2 it follows that $d(u, v, G_i) \leq h'$, which implies that $B(u, G_i, h')$ contains π' . Hence $|B(u, G_i, h')| \geq h$ and $t(u) \leq i$. This shows that $2^{t(u)} \leq 2^i \leq d^h(u, v, G) \leq w(\pi', G) \leq 2\epsilon'w(\pi, G)$. \square

Computing Types of Nodes. To compute $t(u)$ for all nodes u , it is sufficient for every node u to know, for each i , whether $|B(u, G_i, h')| \geq h$. We do this by solving the (S, γ, σ) -detection problem on G_i with $S = V(G)$, $\gamma = h'$ and $\sigma = h$, i.e., we compute the list

$L(u, S, \gamma, \sigma, G)$ for all nodes u , which contains the σ nodes from S that are closest to u , provided their distance is at most γ . By Theorem 2.4 this requires $O(\gamma + \sigma) = O(h + h') = O(\sqrt{n})$ rounds. For any node u , $|L(u, V(G), h', h, G)| = h$ if and only if $|B(u, G_i, h')| \geq h$. Thus, after we solve the (S, γ, σ) -detection problem on all G_i , using $\tilde{O}(\sqrt{n})$ rounds, every node u can compute its type $t(u)$ without any additional computation.

4.1.2 Selecting Centers via Ruling Sets

Having computed the types of the nodes, we compute ruling sets for the nodes of each type to select a small subset of nodes of each type as centers. Remember the two properties of an (α, β) -ruling set T of a base set U : (1) all nodes of T are at least distance α apart and (2) each node in $U \setminus T$ has at least one “ruling” node of T in distance β . We use the algorithm of Theorem 2.7 to compute a $(2h' + 1, (2h' + 1)\lambda)$ -ruling set T_i for each graph G_i where the input set U_i consists of all nodes of type i . The number of rounds for this computation is $\tilde{O}(h') = \tilde{O}(\sqrt{n})$. We define the set of centers as $V' = (\bigcup_i T_i) \cup \{s\}$. Property (1) allows us to bound the number of centers and by property (2) the centers “almost” hit all paths with \sqrt{n} edges.

Lemma 4.3. (1) The number of centers is $|V'| = \tilde{O}(\sqrt{n}/\epsilon)$. (2) For any path π containing exactly \sqrt{n} edges, there is a node u in π and a center $v \in V'$ such that $d^{h^*}(u, v, G) = \epsilon w(\pi, G)$, where $h^* = 9\sqrt{n}\lambda$.

Proof. (1) For each i , consider any two nodes u and v in T_i . Since $d(u, v, G_i) > 2h'$ by Property (1) of the ruling set, $B(u, G_i, h') \cap B(v, G_i, h') = \emptyset$. As every node $u \in T_i$ is of type i , $|B(u, G_i, h')| \geq h$ for every $u \in T_i$. We can therefore uniquely assign h nodes to every node $u \in T_i$ and thus $|T_i| \leq n/h = O(\sqrt{n}/\epsilon)$.

(2) By Lemma 4.2, there is a node u in π such that $2^{t(u)} = 2\epsilon'w(\pi, G)$. Moreover, there is a center v in the ruling set $T_{t(u)}$ such that $d(u, v, G_{t(u)}) \leq (2h' + 1)\lambda \leq 3h'\lambda \leq h^*$. (The last inequality is because $h' \leq 3\sqrt{n}$.) Let π' be the shortest path between u and v in $G_{t(u)}$. Then $w(\pi', G_{t(u)}) = d(u, v, G_{t(u)}) \leq h^*$, and as a consequence π' contains at most h^* edges. It follows that

$$\begin{aligned}
d^{h^*}(u, v, G) &\leq w(\pi', G) \\
&= \sum_{(x,y) \in E(\pi')} w(x, y, G) \\
&\leq \sum_{(x,y) \in E(\pi')} \rho_{t(u)} \cdot w(x, y, G_{t(u)}) \quad (\text{recall that } w(x, y, G_{t(u)}) = \lceil \frac{w(x,y,G)}{\rho_{t(u)}} \rceil) \\
&\leq \rho_{t(u)} \cdot w(\pi', G_{t(u)}) \\
&= \rho_{t(u)} \cdot d(u, v, G_{t(u)}) \\
&= \frac{\epsilon' 2^{t(u)}}{h} d(u, v, G_{t(u)}) \\
&\leq \frac{\epsilon' 2^{t(u)}}{h} 3h'\lambda \\
&= 9\lambda 2^{t(u)} \\
&\leq 18\lambda \epsilon' w(\pi, G)
\end{aligned}$$

$$= \epsilon w(\pi, G)). \quad \square$$

4.1.3 Computing Distances to Centers

Recall that we use parameters defined above Section 4.1.1. Let $k = 2h^* + 2\sqrt{n}$, where $h^* = 9\sqrt{n}\lambda$ (as in Lemma 4.3). In this step, we compute for every node u and every center v a value $\hat{d}(u, v)$ that is a $(1 + \epsilon)$ -approximation of $d^k(u, v, G)$ such that each node u knows $\hat{d}(u, v)$ for all centers v . In particular we also compute $\hat{d}(u, v)$ for all pairs of centers u and v . To do this we follow the idea of partial distance estimation [LP15]. As in Section 4.1.1, we do this by solving the source detection problem on a graph with rounded weights.¹² For any integer i , let $\varphi_i = \frac{\epsilon^2 i}{k}$. Let \hat{G}_i be the weighted graph such that $w(u, v, \hat{G}_i) = \lceil \frac{w(u, v, G)}{\varphi_i} \rceil$ for every edge (u, v) in G .

We solve the (S, γ, σ) -detection problem on \hat{G}_i for all $0 \leq i \leq \lfloor \log nW \rfloor$, with parameters $S = V'$, $\gamma = k'$, and $\sigma = |V'|$ (where $|V'| = \tilde{O}(\sqrt{n}/\epsilon)$ by Lemma 4.3). Using the algorithm of Theorem 2.4 repeatedly this takes $\tilde{O}(\gamma + \sigma) = \tilde{O}(\sqrt{n})$ rounds. At termination, every node u knows the distances up to k' to all centers in all \hat{G}_i ; i.e., it knows $d(u, v, k', \hat{G}_i)$ for all i and all centers v . For every node u and center v we set $\hat{d}(u, v) = \min_i \{\varphi_i \cdot d(u, v, k', \hat{G}_i)\}$. Every node u can compute $\hat{d}(u, v)$ without any additional communication as soon as the source detection algorithm is finished. Now, observe that for the index i^* such that $2^{i^*} \leq d^h(u, v, G) \leq 2^{i^*+1}$, it follows from Equation (3) of Lemma 2.2 that $d(u, v, G_{i^*}) \leq k'$ which implies that $d(u, v, k', G_{i^*}) = d(u, v, G_{i^*})$. With Equations (2) and (4) it then follows that

$$d(u, v, G) \leq \hat{d}(u, v) \leq \varphi_{i^*} \cdot d(u, v, k', G_{i^*}) = \varphi_{i^*} \cdot d(u, v, G_{i^*}) \leq (1 + \epsilon)d^k(u, v, G).$$

Hence $\hat{d}(u, v)$ is the desired $(1 + \epsilon)$ -approximation of $d^k(u, v, G)$.

4.1.4 Completing the Proof of Theorem 4.1

We define our final overlay network to be the graph G' where the weight between any two centers $u, v \in V(G')$ is $\hat{d}(u, v)$ (as computed in Section 4.1.3). Additionally, for every node $u \in V(G)$ we store the value of $\hat{d}(u, v)$ to all centers $v \in V(G')$. We now show that all properties stated in Theorem 4.1 hold for G' . Since we need $\tilde{O}(\sqrt{n})$ rounds in Sections 4.1.1 and 4.1.3 and $\tilde{O}(\sqrt{n})$ rounds in Section 4.1.2, the running time to construct G' is $\tilde{O}(\sqrt{n})$. Moreover, $|V(G')| = \tilde{O}(\sqrt{n})$ as shown in Lemma 4.3. This is as claimed in the first part of Theorem 4.1. It is thus left to prove the following statement in Theorem 4.1: “for every node $u \in V(G)$, as soon as u receives a $(1 + \epsilon)$ -approximation $\tilde{d}(s, v)$ of $d(s, v, G')$ for all centers $v \in V(G')$, it can infer a $(1 + 7\epsilon)$ -approximate value of $d(u, s, G)$ without any additional communication.”

Consider any node u and let π be the shortest path between s and u in G . If π contains less than \sqrt{n} edges, then $d^k(u, s, G) = d(u, s, G)$ and thus the value $\hat{d}(u, s)$ known by u is already a $(1 + \epsilon)$ -approximate value of $d(u, s, G)$. If π contains at least \sqrt{n} edges, then partition π into subpaths $\pi_0, \pi_1, \dots, \pi_t$ (for some t) where π_0 contains s , π_t contains u , π_0 contains at most \sqrt{n} edges, and every subpath except π_0 contains exactly \sqrt{n} edges. By

¹²We note that the algorithm and analysis described in this subsection is essentially the same as the proof of [LP15, Theorem 3.3]. We cannot use the result in [LP15] directly since we need a slightly stronger guarantee, which can already be achieved by the same proof. (We thank Christoph Lenzen for a communication regarding this.)

Lemma 4.3, for every $1 \leq i \leq t$, there is a node u_i and a center v_i such that (i) u_i is in π_i and (ii) $d^{h^*}(u_i, v_i, G) = \epsilon w(\pi_i, G)$. Additionally since u_i and u_{i+1} lie on π , their shortest path is the subpath of π between them and, thus, it consists of at most $2\sqrt{n}$ edges. It follows that $d^{2\sqrt{n}}(u_i, u_{i+1}, G) = d(u_i, u_{i+1}, G)$. By our choice of $k = 2h^* + 2\sqrt{n}$ the triangle inequality gives

$$\begin{aligned} d^k(v_i, v_{i+1}, G) &\leq d^{h^*}(v_i, u_i, G) + d^{2\sqrt{n}}(u_i, u_{i+1}, G) + d^{h^*}(u_{i+1}, v_{i+1}, G) \\ &\leq d(u_i, u_{i+1}, G) + \epsilon w(\pi_i, G) + w(\pi_{i+1}, G). \end{aligned}$$

By the same argument, $d^k(s, v_1, G) \leq d(s, u_1, G) + \epsilon w(\pi_1, G)$ and $d^k(u, v_t, G) \leq d(u, u_t, G) + \epsilon w(\pi_t, G)$. As every edge (x, y) in G' has weight $\hat{d}(x, y)$, we get

$$\begin{aligned} \tilde{d}(s, v_t) + \hat{d}(u, v_t) &\leq (1 + \epsilon)d(s, v_t, G') + \hat{d}(u, v_t) \\ &\leq (1 + \epsilon) \left(d(s, v_1, G') + \sum_{i=1}^{t-1} d(v_i, v_{i+1}, G') \right) + \hat{d}(u, v_t) \\ &\leq (1 + \epsilon) \left(\hat{d}(s, v_1) + \sum_{i=1}^{t-1} \hat{d}(v_i, v_{i+1}) \right) + \hat{d}(u, v_t) \\ &\leq (1 + \epsilon)^2 \left(d^k(s, v_1, G) + \sum_{i=1}^{t-1} d^k(v_i, v_{i+1}, G) \right) + (1 + \epsilon)d^k(u, v_t) \\ &\leq (1 + \epsilon)^2 \left(d^k(s, v_1, G) + \sum_{i=1}^{t-1} d^k(v_i, v_{i+1}, G) + d^k(u, v_t, G) \right) \\ &\leq (1 + \epsilon)^2 \left(d(s, u, G) + \epsilon \sum_{i=0}^t w(\pi_i, G) \right) \\ &= (1 + \epsilon)^2 (d(s, u, G) + \epsilon d(s, u, G)) \\ &\leq (1 + \epsilon)^3 d(s, u, G) \\ &\leq (1 + 7\epsilon) d(s, u, G). \end{aligned}$$

Thus, when u receives $\tilde{d}(s, v')$ for all centers $v \in V(G')$, it can compute $\min_{v \in V(G')} (\tilde{d}(s, v) + \hat{d}(u, v))$, which, as we just argued, is a $(1 + 7\epsilon)$ -approximation of $d(s, u, G)$.

4.2 Computing a Hop Set on an Overlay Network

We now show how to simulate the hop set algorithm presented in Section 3 on an overlay network G' and how to compute approximate shortest paths from s in G' using the hop set. Throughout the algorithm we will work on overlay networks whose node set is the set of centers $V(G')$, but which might have different edge weights as, e.g., Procedure 4 calls Procedure 3 on overlay networks with modified edge weights. Thus, we will use G'' to refer to an overlay network on which Procedures 2 to 5 run to emphasize the fact that they might not equal G' . We let N be the number of centers in G' and G'' . Thus $N = \tilde{\Theta}(n^{1/2})$.

4.2.1 Computing Bounded-Distance Single-Source Shortest Paths

We will repeatedly use an algorithm for computing a shortest-path tree up to distance R rooted at s on an overlay network G'' , where $R = O(n^{o(1)})$. At the end of the algorithm every

node knows this tree. We do this in a breadth-first search manner, in $R + 1$ iterations. Like in Dijkstra's algorithm, every node keeps a tentative distance $\delta(s, u)$ from s and a tentative parent in the shortest-path tree, i.e., a node v such that $\delta(s, u) = \delta(s, v) + w(u, v, G'')$. Initially, $\delta(s, s) = 0$ and $\delta(s, v) = \infty$ for every node $v \neq s$. In the L^{th} iteration, for L from 0 up to R , all nodes in G'' whose tentative distance $\delta(s, u)$ is exactly L broadcast¹³ to all other nodes a message $(u, \delta(s, u), v)$ where v is the parent of u . Using this information, every node u will update ("relax") its tentative distance $\delta(s, u)$ and its tentative parent.

Clearly, after the L^{th} iteration, centers that have distance $L + 1$ from s (i.e., that are at level L in the shortest-path tree) will already know their correct distance. Thus, at the end of the last iteration every center knows the shortest-path tree rooted at s up to distance R in G'' . To analyze the running time, note that over R rounds we broadcast N messages in total, and if m_L messages are broadcast in the L^{th} iteration, then this iteration takes $O(m_L + D)$ rounds. (We emphasize that the number of rounds depends on the diameter D of the original network, and not of G'' .) The total number of communication rounds used over all iterations is thus $O(RD + \sum_L m_L) = O(RD + N)$.

4.2.2 Computing Priorities

We will simulate Procedure 1. All necessary parameters can be computed beforehand and thus do not require any communication. Initially every center knows that it is contained in $A_0 = V$. To compute A_{i+1} given that A_i is known (i.e., every center knows whether it is in A_i or not), we compute the list $L(v, A_i, R, q, G'')$ for every center v using a source detection algorithm and distribute each list to every center. Then every center runs the same deterministic greedy hitting set approximation algorithm to compute A_{i+1} . We will obtain $(A_i)_{0 \leq i \leq p}$ by repeating this for p iterations. Thus, we only have to solve the (S, γ, σ) -source detection problem with $S = A_i$, $\gamma = R$, and $\sigma = q$ on an overlay network G'' . For this purpose we simulate the source detection algorithm of Roditty, Thorup, and Zwick [RTZ05] (see Theorem 2.5).¹⁴

Recall that in the first phase of this algorithm, we add an artificial source s^* to G with a 0-length edge between s^* and every center in S and perform a single-source shortest paths computation from s^* up to distance γ . We do this in the same way as the bounded-distance shortest-path tree algorithm described in Section 4.2.1, except that we initially set $\delta(s^*, u) = 0$ for every center $u \in S$. The subsequent iterations in this phase are exactly as described in Section 4.2.1. Note that due to the broadcast messages every center knows that shortest-path tree from s^* in G'' at the end of the first phase. In the next phase, we have to compute a shortest-path tree on the graph G''_1 . We observe the following from the construction of G''_1 described in Section 3.1.

Observation 4.4. *Every center u in G'' will know all edges incident to it in G''_1 if it knows, for every center v in G'' , (1) $U_1(v)$ and (2) the distance (in G'') from v to all centers in*

¹³More precisely, there is a designated node (e.g. the node with lowest ID) that aggregates and distributes the messages (via upcasting and downcasting on the breadth-first search tree of the network G), and tells other nodes when the iteration starts and ends.

¹⁴Although the algorithm of Lenzen and Peleg [LP13] (see Theorem 2.4) was designed for the distributed setting, we do not know how to simulate it on the overlay network with the running time guarantees we require. In particular we have no guarantee that in $O(D)$ rounds each node of the overlay network can send a message to all its neighbors, which is allowed in one round of the original algorithm. The reason is the congestion we obtain by individual nodes being part of many paths used for hop set edges.

$U_1(v)$. In particular, it will know this information if it knows the shortest-path tree obtained in the previous phase.

Since the shortest-path tree obtained in the previous phase is known to every center, we can assume that every center u in G'' knows all its incident edges in G''_1 at the beginning of the phase. Now we compute the shortest-path tree up to distance R in a similar manner as in Section 4.2.1. The only exceptions are that (1) initially every center u in G'' knows that \bar{v} has distance 0 to s^* , for every center \bar{v} in G''_1 , and they use this information to initialize the value of $\delta(s^*, u)$ and (2) that they use the adjacent edges in G''_1 and *not* in G'' to update their distance to s^* .

The subsequent phases are simulated in the same manner; i.e., using the shortest-path tree computed in the previous phases, every center knows its incident edges in G''_j . This allows us to compute the shortest-path tree in G''_j in the breadth-first search manner. After we finish all σ phases, every center in G'' will know $L(v, S, \gamma, \sigma, G'')$ for all v . Using the aforementioned analysis of the bounded-distance shortest-path tree, it follows that every phase takes $O(\gamma D + N)$ rounds and as there are σ rounds the whole source detection algorithm takes $O(\sigma \gamma D + \sigma N)$ rounds. For computing the priorities we repeat this for all p priorities and thus get the following.

Lemma 4.5. *The above algorithm deterministically computes a hierarchy $(A_i)_{0 \leq i \leq p}$ as in Lemma 3.2 in $O(pq(RD + N))$ rounds. With our choice of parameters ($N = \tilde{O}(n^{1/2})$, $p \leq \log n$, $q = O(N^{o(1)})$, and $R = O(N^{o(1)})$), the running time of the above algorithm is $O(n^{1/2+o(1)} + Dn^{o(1)})$.*

4.2.3 Computing Clusters

We now describe how to compute clusters in G'' . At the end of the cluster computation every center will know the cluster $C(u)$ of every center u . We do this by simulating Procedure 2. First, we need to compute $d(v, A_{i+1}, R, G)$, for every $1 \leq i \leq p$. This can be done in exactly the same way as in the first phase of computing the hierarchy $(A_i)_i$; i.e., we simply add a virtual source s^* and edges of weight zero between s^* and centers in A_i , and compute the shortest-path tree up to distance R rooted at s^* . Since such a tree can be computed in $\tilde{O}(RD + N)$ rounds and we have to compute $p \leq \log n$ such trees, the total time we need here is $\tilde{O}(RD + N)$.

Next, we compute a shortest-path tree up to distance R from every center u in G'' , as described in Procedure 2. That is, at Iteration L (starting with $L = 0$ and ending with $L = R$), every center v having (a) $\delta(u, v)$ (the tentative distance from u to v) equal to L and (b) additionally $\delta(u, v) < d(v, A_{i+1}, R, G)$ will broadcast¹⁵ its distance to u to all other centers so that every other center, say w , can (1) update its tentative distance $\delta(u, w)$ and (2) add v and $\delta(u, v)$ to its locally stored copy of $C(u)$. It thus follows that for every center u the number of messages broadcast during the computation of $C(u)$ is $|C(u)|$. Thus, there are $\sum_{u \in V(G'')} |C(u)|$ messages broadcast in total, which is bounded from above by $\tilde{O}(pN^{1/p}) = O(n^{1/2+o(1)})$ due to Theorem 3.3.

Note that this procedure computes $C(v)$, for all centers v , *in parallel*. Each Iteration L requires $O(D + \sum_{v \in V(G'')} m_{v,L})$ rounds, where $m_{v,L}$ is the number of messages needed to be

¹⁵We note again that to do this, there is a designated center that aggregates and distributes the messages (via upcasting and downcasting), and tells other centers when the iteration starts and ends.

broadcast at Iteration L in order to compute $C(v)$. The total number of rounds over all R iterations is thus

$$O(RD + \sum_{0 \leq L \leq R} \sum_{v \in V(G'')} m_{v,L}) = O(RD + \sum_{v \in V(G'')} |C(v)|) = O(n^{o(1)}D + n^{1/2+o(1)}). \quad (10)$$

Note that since the computation is done by broadcasting messages, every center knows cluster $C(v)$ for all v at the end of this computation.

Lemma 4.6. *For any overlay network G'' with $\tilde{O}(n^{1/2})$ centers, the above algorithm deterministically computes clusters up to distance $R = n^{o(1)}$ (where every center knows $C(u)$ for all centers u) that satisfies Theorem 3.3 in $O(n^{o(1)}D + n^{1/2+o(1)})$ rounds.*

4.2.4 Computing the Hop Reduction with Additive Error

We will simulate Procedure 3. All necessary parameters can be computed beforehand and thus do not require any communication. We can then execute $\text{CLUSTERS}(G, p, R)$ using the above algorithm to get $(C(v), \delta(v, \cdot))_{v \in V}$. With this information the set F can be computed without any additional communication. Thus, executing $\text{CLUSTERS}(G, p, R)$ is the only part for computing F that requires communication. By Lemma 4.6 the total time needed to execute Procedure 3 is therefore

$$O(n^{o(1)}D + n^{1/2+o(1)}). \quad (11)$$

4.2.5 Computing the Hop Reduction without Additive Error

We will simulate Procedure 4. All necessary parameters can be computed beforehand and thus do not require any communication. Moreover, every center knows about the edges incident to it and can thus compute \hat{G}'_r by scaling down edge weights without any communication. We then execute Procedure 3 to compute \hat{F}'_r . Knowing \hat{F}'_r we can compute F without any additional communication. Thus, executing Procedure 3 is the only part for computing F that requires communication and it is executed $O(\log nW)$ times. As simulating Procedure 3 takes time $O(n^{o(1)}D + n^{1/2+o(1)})$, as argued above (cf. Equation (11)), the total time needed to execute Procedure 4 is

$$O(n^{o(1)}D \log(nW) + n^{1/2+o(1)} \log(nW)). \quad (12)$$

4.2.6 Computing the Hop Set

We will simulate Procedure 5. All necessary parameters can be computed beforehand. Computing F_{i+1} is done by simulating Procedure 4 on the graph H_i , which, as argued above (cf. Equation (12)), takes time $O(n^{o(1)}D \log(nW) + n^{1/2+o(1)} \log(nW))$. As every center knows its incident edges, the graph H_{i+1} can be computed from F_{i+1} without any additional communication. As we execute Procedure 4 $p \leq \log n$ times, the total time needed to simulate Procedure 5 is

$$O(pn^{o(1)}D \log(nW) + pn^{1/2+o(1)} \log(nW)) = O(n^{o(1)}D \log(nW) + n^{1/2+o(1)} \log(nW)).$$

By running this algorithm on G' (which, as pointed out, involves performing hop reductions and computing clusters on some other overlay networks), we obtain the following theorem.

Theorem 4.7. *There is a deterministic algorithm that computes a hop set of G' in $O(n^{o(1)}D \log(nW) + n^{1/2+o(1)} \log(nW))$ rounds. In the end of the process, every center knows every edge in the hop set.*

4.2.7 Routing via the Hop Set

Remember that the overlay network is computed using the source detection algorithm of Lenzen and Peleg [LP13]. If a node x of the overlay network wants to send a message to one of its neighbors y in the overlay network it can do so by routing the message along a path in the original network whose length is upper-bounded by the weight of the overlay edge (x, y) . This routing can be obtained by modifying the source detection algorithm to additionally construct BFS trees rooted at the sources (see [LP13]), which in our case are the nodes of the overlay network.

When we compute the hop set on the overlay network we broadcast all computed clusters to all nodes in the network. In this way the clusters, the corresponding partial shortest path trees of the clusters, as well as the hop set edges become global knowledge. Therefore every node in the overlay network learns for every hop set edge (x, y) to which path from x to y in the overlay network it corresponds. Thus, also for every hop set edge (x, y) of the overlay network, x can send a message to y by routing the message along a path in the original network whose length is upper-bounded by the weight of the overlay edge (x, y) . This means that the hop set computed by our algorithm has the following *awareness* property, as introduced in the full version of [EN16b]: A hop set F for a graph G is called *aware* if for every hop-set edge $(x, y) \in F$ of weight b there exists a corresponding path π in G between x and y of length b . Furthermore, every node v on π knows $d_\pi(v, x)$ and $d_\pi(v, y)$, and its neighbors on π .

4.3 Final Steps

Let H be the graph obtained by adding to G' the edges of the $(n^{o(1)}, o(1))$ -hop set computed above. To $(1 + o(1))$ -approximate $d(s, v, G')$ for every center v in G' , it is sufficient to $(1 + o(1))$ -approximate the h -hop distance $d^h(s, v, H)$, for some $h = n^{o(1)}$. The latter task can be done in $O(hD + |V(G')|) = O(n^{o(1)}D + n^{1/2+o(1)})$ rounds by the same method as in Lemma 4.6 in the full version of [Nan14]. We give a sketch here for completeness. Let $\epsilon = 1/\log n$. For any $1 \leq i \leq \log(nW)$, let H'_i be the graph obtained by rounding edge weights in H' as in Section 4.1.1; i.e., for every edge (u, v) we set $w(u, v, H'_i) = \lceil \frac{w(u, v, H')}{\rho_i} \rceil$, where $\rho_i = \frac{\epsilon^{2^i}}{h}$. For each H_i , we compute the shortest-path tree rooted at s up to distance $R = O(h/\epsilon)$, which can be done in $O(RD + n^{1/2+o(1)}) = O(n^{o(1)}D + n^{1/2+o(1)})$ rounds, using the procedure described in Section 4.2. This gives $d(s, v, R, H'_i)$ for every center v . We then use the following value as $(1 + o(1))$ -approximation of $d^h(s, v, H)$ (and thus of $d(s, v, G')$): $\tilde{d}(s, v) = \min_i \rho_i d(s, v, R, H'_i)$. The correctness of this algorithm follows from Lemma 2.2.

Once we have $(1 + o(1))$ -approximate values of $d(s, v, G')$ for every center $v \in V(G')$, we can broadcast these values to the whole network in $\tilde{O}(D + n^{1/2})$ rounds. Theorem 4.1 then implies that we have a $(1 + o(1))$ -approximate solution to the single-source shortest paths problem on the original network. The total time spent is $O(n^{o(1)}D + n^{1/2+o(1)})$. By observing that the term $n^{o(1)}D$ will show up in the running time only when $D = \omega(n^{o(1)})$, we can write the running time as $O(D^{1+o(1)} + n^{1/2+o(1)})$, as claimed in the beginning.

We thus have obtained the following result.

Theorem 4.8. *There is a deterministic distributed algorithm that, on any weighted undirected network, computes $(1 + o(1))$ -approximate shortest paths between a given source node s and every other node in $O(n^{1/2+o(1)} + D^{1+o(1)})$ rounds.*

5 Algorithms on Other Settings

5.1 Congested Clique

In the congested clique model, the underlying communication network is a complete graph. Thus, in each round every node can send a message to every other node. Apart from this topological constraint, the congested clique is similar to the CONGEST model.

We compute an $(n^{o(1)}, o(1))$ -hop set on a congested clique by simulating the hop set construction algorithm in the same way as on an overlay network as presented in Section 4.2. (However, we do *not* compute an overlay network here.) The only difference is the number of rounds needed for nodes to broadcast messages to all other nodes. Consider the situation that m' messages are to be broadcast by some nodes. On a network of arbitrary topology, we will need $O(D + m')$ rounds. On a congested clique, however, we only need $O(m'/n)$ rounds using the routing scheme of Dolev et al. [DLP12, Lemma 1] (also see [Len13]): If each node is source and destination of up to n messages of size $O(\log n)$ (initially only the sources know destinations and contents of their messages), we will need $O(1)$ rounds to route the messages to their destinations. In particular, we can broadcast n messages in $O(1)$ rounds, and thus m' messages in $O(m'/n)$ rounds. Using this fact, the number of rounds needed for the algorithm in Section 4.2 reduces from $O(RD + \sum_{v \in V(G'')} |C(v)|)$ (cf. Equation (10)) to $O(R + \sum_{v \in V(G'')} |C(v)|/n) = \tilde{O}(R + pn^{1/p}) = O(n^{o(1)})$ on a congested clique.¹⁶

Once we have an $(n^{o(1)}, o(1))$ -hop set, we proceed as in Section 4.3. Let H be the graph obtained by adding to the input graph G the edges of the $(n^{o(1)}, o(1))$ -hop set. We can treat H as a congested clique network with different edge weights from G . (H can be computed without any communication since every node already knows the hop set.) To $(1 + o(1))$ -approximate $d(s, v, G)$ for every node v in G , it is sufficient to compute the h -hop distance $d^h(s, v, H)$, for some $h = n^{o(1)}$. To do this, we follow the same approach for this problem as in [Nan14, Section 5.1], where we execute the distributed version of the Bellman-Ford algorithm for h rounds. That is, every node u maintains a tentative distance from the source s , denoted by $\delta(s, u)$ and in each round every node u broadcasts $\delta(s, u)$ to all other nodes. It can be shown that after k rounds every node v knows the k -hop distance (i.e., $\delta(s, u) = d^k(s, v, H)$) correctly, and thus after h rounds we will get the h -hop distances as desired.¹⁷

¹⁶Instead of relying on the result of Dolev et al., we can use the following algorithm to broadcast m' messages in $O(m'/n)$ rounds as follows. We assign an order to the messages, where messages sent by a node with smaller ID appears first in the order and messages sent by the same node appear in any order (a node can learn the order of its messages after it knows how many messages other nodes have). We then broadcast the first n messages according to this order, say M_1, \dots, M_n , where message M_i is sent to a node with the i^{th} smallest ID, and such a node sends M_i to all other nodes. This takes only two rounds. The next messages are handled similarly. This algorithm broadcasts each n messages using 2 rounds, and thus the total number of rounds is $O(m'/n)$.

¹⁷Note that instead of the Bellman-Ford algorithm, one can also follow the steps in Section 4.3 instead.

Theorem 5.1. *There is a deterministic distributed algorithm that, on any weighted congested clique, computes $(1 + o(1))$ -approximate shortest paths between a given source node s and every other node in $O(n^{o(1)})$ rounds.*

5.2 Streaming Algorithm

In the graph streaming model, the edges of the input graph are presented to the algorithm in an arbitrary order. The goal is to design algorithms that process this “stream” of edges using as little space as possible. In the multipass streaming model we are allowed to read the stream several times and want to keep both the number of passes and the amount of space used as small as possible.

Our streaming algorithm for constructing an $(n^{o(1)}, o(1))$ -hop set proceeds in almost the same way as in Section 4.2. First observe that a shortest-path tree up to distance R can be computed in $O(R)$ passes and with $\tilde{O}(n)$ space: We use the space to remember the tentative distances of the nodes to s , and the shortest-path tree computed so far. At the end of the L^{th} pass we add nodes having distances exactly L to the shortest-path tree, and update the distance of their neighbors in the $(L + 1)^{\text{th}}$ pass.

We compute the priorities, as described in Section 4.2.2, the algorithm proceeds in $q = n^{o(1)}$ phases where in the L^{th} phase we compute a shortest-path tree rooted at s^* up to distance $R = n^{o(1)}$ on graph G_{i-1} (where $G_0 = G$). As in 4.4, we can construct G_i if we know the shortest-path tree rooted at s^* up to distance R . Thus, after computing such a tree, the algorithm can construct G_i and proceed to the next phase without reading the stream. The algorithm therefore needs $O(qR) = n^{o(1)}$ passes and $O(qn) = O(n^{1+o(1)})$ space.

To compute clusters, we compute n shortest-path trees up to distance R rooted at different nodes in parallel. The number of passes is clearly $O(R)$. The space is bounded by the sum of the sizes of the shortest-path trees. This is $O(\sum_{v \in V(G'')} |C(v)|)$ which, by Theorem 3.3, is $\tilde{O}(pn^{1+1/p}) = O(n^{1+o(1)})$. To compute the hop set we only have to compute clusters $\tilde{O}(\log W)$ times. So, we need $O(n^{o(1)} \log W)$ passes and $O(n^{1+o(1)} \log W)$ space in total. By considering the edges of the hop set in addition the edges read from the stream it suffices to compute approximate single-source shortest paths up to $n^{o(1)}$ hops. Using the rounding technique this can be done in $O(n^{o(1)})$ additional passes.

Theorem 5.2. *There is a deterministic streaming algorithm that, given any weighted directed graph, computes $(1 + o(1))$ -approximate shortest paths between a given source node s and every other node in $O(n^{o(1)} \log W)$ passes with $O(n^{1+o(1)} \log W)$ space.*

6 Conclusion and Open Problems

We present deterministic distributed $(1 + o(1))$ -approximation algorithms for solving the single-source shortest paths problem on distributed weighted networks and other settings. The efficiencies of our algorithms match the known lower bounds up to an $n^{o(1)}$ factor. Important tools consist of a construction the hop set and a deterministic process that replaces the well-known (randomized) hitting set argument. Our understanding of these tools

This gives a $(1 + o(1))$ -approximate value for $d^h(s, v, H)$ for every node v , which is sufficient for computing a $(1 + o(1))$ -approximate value for $d(s, v, G)$. This algorithm is however more complicated.

is still limited. We leave as a major open problem whether we can improve our understanding of these tools in the following aspects.

First, the main reason that the $n^{o(1)}$ factor shows up in the bounds of our algorithms is because we rely on a $(n^{o(1)}, o(1))$ -hop set of size $O(n^{1+o(1)})$. To use the same approach to get similar bounds but without the $n^{o(1)}$ term, we need efficient algorithms to construct a $(\text{polylog}(n), o(1))$ -hop set of size $\tilde{O}(n)$. However, such hop set is not known to exist. We leaving the existence of such hop set (ignoring the efficiency in constructing it) as the first open question.¹⁸

Secondly, our deterministic replacement of the hitting set argument works only when the input graph is undirected. Our second open problem is thus how to derandomize algorithms on directed graphs (where edge directions do not affect the communication; see [Nan14, Nan16] for more details). In particular, it is known that single-source shortest paths can be $(1 + \epsilon)$ -approximated on directed weighted graphs in $\tilde{O}(n^{1/2}D^{1/2} + D)$ time [Nan14], and single-source reachability can be computed in $\tilde{O}(n^{1/2}D^{1/4} + D)$ time [GU15]. However, these results are obtained by *randomized* algorithms, and whether there are sublinear-time *deterministic* algorithms for these problems is still open.

Finally, while our paper essentially settles the running time for computing single-source shortest paths approximately, the best running time for solving this problem *exactly* is still linear (by the Bellman-Ford algorithm). Whether there is a sublinear algorithm is a major open problem. In fact, in the past few years we have much better understood how to *approximately* solve basic graph problems, such as minimum cut, single-source shortest paths, all-pairs shortest paths, and maximum flows, on distributed networks (e.g. [NS14, GK13, GKK⁺15]). However, when it comes to solving these problems *exactly*, almost nothing is known. Understanding the complexity of exact algorithms is an important open problems.

We refer to [Nan14, Nan16] for further open problems.

Acknowledgment

We thank Michael Elkin, Stephan Friedrichs, and Christoph Lenzen for their comments and questions on the previous version of this paper. D. Nanongkai thanks Michael Elkin for bringing the notion of hop set to his attention.

References

- [ADP80] Giorgio Ausiello, Alessandro D’Atri, and Marco Protasi. “Structure Preserving Reductions among Convex Optimization Problems”. In: *Journal of Computer and System Sciences* 21.1 (1980). Announced at ICALP’77, pp. 136–153 (cit. on p. 12).
- [Bas08] Surender Baswana. “Streaming algorithm for graph spanners—single pass and constant processing time per edge”. In: *Information Processing Letters* 106.3 (2008), pp. 110–114 (cit. on p. 4).

¹⁸We note that recently [BKK⁺16] showed how to eliminate the $n^{o(1)}$ term from our results by bypassing the hop set. Nevertheless, the question whether a $(\text{polylog}(n), o(1))$ -hop set of size $\tilde{O}(n)$ exists is interesting on its own and potentially has other applications.

- [BE13] Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013. ISBN: 9781627050180 (cit. on p. 10).
- [Bel58] Richard Bellman. “On a Routing Problem”. In: *Quarterly of Applied Mathematics* 16.1 (1958), pp. 87–90 (cit. on p. 3).
- [Ber09] Aaron Bernstein. “Fully Dynamic $(2 + \epsilon)$ Approximate All-Pairs Shortest Paths with Fast Query and Close to Linear Update Time”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2009, pp. 693–702 (cit. on pp. 6, 8, 15).
- [Ber16] Aaron Bernstein. “Maintaining Shortest Paths Under Deletions in Weighted Directed Graphs”. In: *SIAM Journal on Computing* 45.2 (2016). Announced at STOC’13, pp. 548–574 (cit. on pp. 5, 8).
- [BHS07] Surender Baswana, Ramesh Hariharan, and Sandeep Sen. “Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths”. In: *Journal of Algorithms* 62.2 (2007). Announced at STOC’02, pp. 74–92 (cit. on p. 5).
- [BKK⁺16] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. “Approximate Undirected Transshipment and Shortest Paths via Gradient Descent”. In: *CoRR* abs/1607.05127 (2016) (cit. on pp. 6, 35).
- [CKK⁺15] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. “Algebraic Methods in the Congested Clique”. In: *Symposium on Principles of Distributed Computing (PODC)*. 2015, pp. 143–152 (cit. on p. 4).
- [Coh00] Edith Cohen. “Polylog-Time and Near-Linear Work Approximation Scheme for Undirected Shortest Paths”. In: *Journal of the ACM* 47.1 (2000). Announced at STOC’94, pp. 132–166 (cit. on p. 5).
- [Coh98] Edith Cohen. “Fast Algorithms for Constructing t -Spanners and Paths with Stretch t ”. In: *SIAM Journal on Computing* 28.1 (1998). Announced at FOCS’93, pp. 210–236 (cit. on p. 8).
- [DFI05] Camil Demetrescu, Irene Finocchi, and Giuseppe F. Italiano. “Handbook on Data Structures and Applications”. In: CRC Press, 2005. Chap. 36: Dynamic Graphs (cit. on pp. 5, 24).
- [DFR09] Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. “Trading Off Space for Passes in Graph Streaming Problems”. In: *ACM Transactions on Algorithms* 6.1 (2009). Announced at SODA’06 (cit. on p. 5).
- [DHK⁺12] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. “Distributed Verification and Hardness of Distributed Approximation”. In: *SIAM Journal on Computing* 41.5 (2012). Announced at STOC’11, pp. 1235–1265 (cit. on pp. 1, 3, 4).

- [DI06] Camil Demetrescu and Giuseppe F. Italiano. “Fully dynamic all pairs shortest paths with real edge weights”. In: *Journal of Computer and System Sciences* 72.5 (2006). Announced at FOCS’01, pp. 813–837 (cit. on p. 5).
- [DLP12] Danny Dolev, Christoph Lenzen, and Shir Peled. ““Tri, Tri Again”: Finding Triangles and Small Subgraphs in a Distributed Setting”. In: *International Symposium on Distributed Computing (DISC)*. 2012, pp. 195–209 (cit. on p. 33).
- [DSDP15] Atish Das Sarma, Michael Dinitz, and Gopal Pandurangan. “Efficient distributed computation of distance sketches in networks”. In: *Distributed Computing* 28.5 (2015). Announced at SPAA’12, pp. 309–320 (cit. on p. 6).
- [EKN⁺14] Michael Elkin, Hartmut Klauck, Danupon Nanongkai, and Gopal Pandurangan. “Can Quantum Communication Speed Up Distributed Computation?” In: *Symposium on Principles of Distributed Computing (PODC)*. 2014, pp. 166–175 (cit. on p. 3).
- [Elk04] Michael Elkin. “Distributed approximation: a survey”. In: *SIGACT News* 35.4 (2004), pp. 40–57 (cit. on pp. 1, 3).
- [Elk06] Michael Elkin. “An Unconditional Lower Bound on the Time-Approximation Trade-off for the Distributed Minimum Spanning Tree Problem”. In: *SIAM Journal on Computing* 36.2 (2006). Announced at STOC’04, pp. 433–456 (cit. on p. 3).
- [Elk11] Michael Elkin. “Streaming and Fully Dynamic Centralized Algorithms for Constructing and Maintaining Sparse Spanners”. In: *ACM Transactions on Algorithms* 7.2 (2011). Announced at ICALP’07, 20:1–20:17 (cit. on p. 4).
- [EN16a] Michael Elkin and Ofer Neiman. “Hopsets with Constant Hopbound, and Applications to Approximate Shortest Paths”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2016 (cit. on p. 6).
- [EN16b] Michael Elkin and Ofer Neiman. “On Efficient Distributed Construction of Near Optimal Routing Schemes: Extended Abstract”. In: *Symposium on Principles of Distributed Computing (PODC)*. 2016, pp. 235–244 (cit. on pp. 6, 32).
- [EZ06] Michael Elkin and Jian Zhang. “Efficient algorithms for constructing $(1 + \epsilon, \beta)$ -spanners in the distributed and streaming models”. In: *Distributed Computing* 18.5 (2006), pp. 375–385 (cit. on p. 4).
- [FKM⁺05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. “On graph problems in a semi-streaming model”. In: *Theoretical Computer Science* 348.2-3 (2005). Announced at ICALP’04, pp. 207–216 (cit. on p. 4).
- [FKM⁺08] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. “Graph Distances in the Data-Stream Model”. In: *SIAM Journal on Computing* 38.5 (2008). Announced at SODA’05, pp. 1709–1727 (cit. on p. 4).

- [FL16] Stephan Friedrichs and Christoph Lenzen. “Parallel Metric Tree Embedding based on an Algebraic View on Moore-Bellman-Ford”. In: *Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 2016, pp. 455–466 (cit. on p. 6).
- [For56] Lester R. Ford. *Network Flow Theory*. Tech. rep. P-923. The Rand Corporation, 1956 (cit. on p. 3).
- [GK13] Mohsen Ghaffari and Fabian Kuhn. “Distributed Minimum Cut Approximation”. In: *International Symposium on Distributed Computing (DISC)*. 2013, pp. 1–15 (cit. on pp. 3, 35).
- [GKK⁺15] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. “Near-Optimal Distributed Maximum Flow: Extended Abstract”. In: *Symposium on Principles of Distributed Computing (PODC)*. 2015, pp. 81–90 (cit. on p. 35).
- [GKP98] Juan A. Garay, Shay Kutten, and David Peleg. “A Sublinear Time Distributed Algorithm for Minimum-Weight Spanning Trees”. In: *SIAM Journal on Computing* 27.1 (1998). Announced at FOCS’93, pp. 302–316 (cit. on p. 3).
- [GO16] Venkatesan Guruswami and Krzysztof Onak. “Superlinear Lower Bounds for Multipass Graph Processing”. In: *Algorithmica* 76.3 (2016). Announced at CCC’13, pp. 654–683 (cit. on p. 4).
- [GPS88] Andrew V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. “Parallel Symmetry-Breaking in Sparse Graphs”. In: *SIAM Journal on Discrete Mathematics* 1.4 (1988). Announced at STOC’87, pp. 434–446 (cit. on pp. 1, 5, 10).
- [GU15] Mohsen Ghaffari and Rajan Udmani. “Brief Announcement: Distributed Single-Source Reachability”. In: *Symposium on Principles of Distributed Computing (PODC)*. 2015, pp. 163–165 (cit. on p. 35).
- [HK95] Monika Henzinger and Valerie King. “Fully Dynamic Biconnectivity and Transitive Closure”. In: *Symposium on Foundations of Computer Science (FOCS)*. 1995, pp. 664–672 (cit. on p. 5).
- [HKN14] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. “Decremental Single-Source Shortest Paths on Undirected Graphs in Near-Linear Total Update Time”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2014, pp. 146–155 (cit. on pp. 6, 15).
- [HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. “A Deterministic Almost-Tight Distributed Algorithm for Approximating Single-Source Shortest Paths”. In: *Symposium on Theory of Computing, (STOC)*. 2016, pp. 489–498 (cit. on p. 6).
- [HP15] Stephan Holzer and Nathan Pinsker. “Approximation of Distances and Shortest Paths in the Broadcast Congest Clique”. In: *International Conference on Principles of Distributed Systems (OPODIS)*. 2015 (cit. on p. 4).

- [HW12] Stephan Holzer and Roger Wattenhofer. “Optimal Distributed All Pairs Shortest Paths and Applications”. In: *Symposium on Principles of Distributed Computing (PODC)*. 2012, pp. 355–364 (cit. on pp. 1, 4).
- [Joh74] David S. Johnson. “Approximation Algorithms for Combinatorial Problems”. In: *Journal of Computer and System Sciences* 9.3 (1974). Announced at STOC’73, pp. 256–278 (cit. on p. 12).
- [Kin99] Valerie King. “Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs”. In: *Symposium on Foundations of Computer Science (FOCS)*. 1999, pp. 81–91 (cit. on p. 5).
- [KKP13] Liah Kor, Amos Korman, and David Peleg. “Tight Bounds for Distributed Minimum-Weight Spanning Tree Verification”. In: *Theory of Computing Systems* 53.2 (2013). Announced at STACS’11, pp. 318–340 (cit. on p. 3).
- [KP08] Maleq Khan and Gopal Pandurangan. “A fast distributed approximation algorithm for minimum spanning trees”. In: *Distributed Computing* 20.6 (2008). Announced at DISC’06, pp. 391–402 (cit. on p. 6).
- [KP98] Shay Kutten and David Peleg. “Fast Distributed Construction of Small k -Dominating Sets and Applications”. In: *Journal of Algorithms* 28.1 (1998). Announced at PODC’95, pp. 40–66 (cit. on p. 3).
- [KS92] Philip N. Klein and Sairam Sairam. “A Parallel Randomized Approximation Scheme for Shortest Paths”. In: *Symposium on Theory of Computing (STOC)*. 1992, pp. 750–758 (cit. on p. 5).
- [Len13] Christoph Lenzen. “Optimal Deterministic Routing and Sorting on the Congested Clique”. In: *Symposium on Principles of Distributed Computing (PODC)*. 2013, pp. 42–50 (cit. on p. 33).
- [LG16] François Le Gall. “Further Algebraic Algorithms in the Congested Clique Model and Applications to Graph-Theoretic Problems”. In: *International Symposium on Distributed Computing (DISC)*. 2016, pp. 57–70 (cit. on p. 4).
- [LP13] Christoph Lenzen and David Peleg. “Efficient Distributed Source Detection with Limited Bandwidth”. In: *Symposium on Principles of Distributed Computing (PODC)*. 2013, pp. 375–382 (cit. on pp. 1, 5, 9, 25, 29, 32).
- [LP15] Christoph Lenzen and Boaz Patt-Shamir. “Fast Partial Distance Estimation and Applications”. In: *Symposium on Principles of Distributed Computing (PODC)*. 2015, pp. 153–162 (cit. on pp. 1, 5, 6, 9, 25, 27).
- [LPS13] Christoph Lenzen and Boaz Patt-Shamir. “Fast Routing Table Construction Using Small Messages”. In: *Symposium on Theory of Computing (STOC)*. 2013, pp. 381–390 (cit. on pp. 1, 3, 5, 23).
- [McG14] Andrew McGregor. “Graph Stream Algorithms: A Survey”. In: *SIGMOD Record* 43.1 (2014), pp. 9–20 (cit. on p. 4).
- [Mađ10] Aleksander Mađry. “Faster Approximation Schemes for Fractional Multicommodity Flow Problems via Dynamic Graph Algorithms”. In: *Symposium on Theory of Computing (STOC)*. 2010, pp. 121–130 (cit. on pp. 5, 8).

- [Nan14] Danupon Nanongkai. “Distributed Approximation Algorithms for Weighted Shortest Paths”. In: *Symposium on Theory of Computing (STOC)*. 2014, pp. 565–573 (cit. on pp. 1, 3–6, 8, 23, 24, 32, 33, 35).
- [Nan16] Danupon Nanongkai. “Some Challenges on Distributed Shortest Paths Problems, A Survey - (Invited Talk)”. In: *SIROCCO*. 2016 (cit. on p. 35).
- [NS14] Danupon Nanongkai and Hsin-Hao Su. “Almost-Tight Distributed Minimum Cut Algorithms”. In: *International Symposium on Distributed Computing (DISC)*. 2014, pp. 439–453 (cit. on pp. 3, 35).
- [Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000. ISBN: 0-89871-464-8 (cit. on pp. 3, 10).
- [PR00] David Peleg and Vitaly Rubinfeld. “A Near-Tight Lower Bound on the Time Complexity of Distributed Minimum-Weight Spanning Tree Construction”. In: *SIAM Journal on Computing* 30.5 (2000). Announced at FOCS’99, pp. 1427–1442 (cit. on p. 3).
- [PT11] David Pritchard and Ramakrishna Thurimella. “Fast Computation of Small Cuts via Cycle Space Sampling”. In: *ACM Transactions on Algorithms* 7.4 (2011). Announced at ICALP’08, 46:1–46:30 (cit. on p. 3).
- [RTZ05] Liam Roditty, Mikkel Thorup, and Uri Zwick. “Deterministic Constructions of Approximate Distance Oracles and Spanners”. In: *International Colloquium on Automata, Languages and Programming (ICALP)*. 2005, pp. 261–272 (cit. on pp. 6, 9, 11–13, 29).
- [RZ11] Liam Roditty and Uri Zwick. “On Dynamic Shortest Paths Problems”. In: *Algorithmica* 61.2 (2011). Announced at ESA’04, pp. 389–401 (cit. on p. 5).
- [San05] Piotr Sankowski. “Subquadratic Algorithm for Dynamic Shortest Distances”. In: *International Computing and Combinatorics Conference (COCOON)*. 2005, pp. 461–470 (cit. on p. 5).
- [Som14] Christian Sommer. “Shortest-Path Queries in Static Networks”. In: *ACM Computing Surveys* 46.4 (2014), 45:1–45:31 (cit. on p. 23).
- [Sub] *List of Open Problems in Sublinear Algorithms: Problem 14*. <http://sublinear.info/14> (cit. on pp. 1, 4).
- [Thu97] Ramakrishna Thurimella. “Sub-Linear Distributed Algorithms for Sparse Certificates and Biconnected Components”. In: *Journal of Algorithms* 23.1 (1997). Announced at PODC’95, pp. 160–179 (cit. on p. 3).
- [Tse15] Lewis Tseng. “PODC 2015 Review”. In: *SIGACT News* 46.4 (2015), pp. 94–102 (cit. on p. 6).
- [TZ05] Mikkel Thorup and Uri Zwick. “Approximate Distance Oracles”. In: *Journal of the ACM* 52.1 (2005). Announced at STOC’01, pp. 74–92 (cit. on pp. 6, 11).
- [TZ06] Mikkel Thorup and Uri Zwick. “Spanners and emulators with sublinear distance errors”. In: *SODA*. 2006, pp. 802–809 (cit. on pp. 6, 11, 15).

- [UY91] Jeffrey D. Ullman and Mihalis Yannakakis. “[High-Probability Parallel Transitive-Closure Algorithms](#)”. In: *SIAM Journal on Computing* 20.1 (1991). Announced at SPAA’90, pp. 100–125 (cit. on pp. [5](#), [24](#)).
- [Zwi02] Uri Zwick. “[All Pairs Shortest Paths using Bridging Sets and Rectangular Matrix Multiplication](#)”. In: *Journal of the ACM* 49.3 (2002). Announced at FOCS’98, pp. 289–317 (cit. on pp. [5](#), [8](#)).

A Proof of Lemma 2.2

To prove Inequality (2), let π_i be a shortest path between u and v in G_i . Observe that if we consider this path in G (with the corresponding edge weights), then its total weight is at most the distance between u and v in G , i.e. $w(\pi_i, G) \geq d(u, v, G)$, because no path in G can have weight less than the shortest path in G . We therefore get

$$\begin{aligned} \rho_i \cdot d(u, v, G_i) &= \rho_i \cdot \sum_{(x,y) \in \pi_i} w(x, y, G_i) = \sum_{(x,y) \in \pi_i} \rho_i \cdot \left\lceil \frac{w(x, y, G)}{\rho_i} \right\rceil \\ &\geq \sum_{(x,y) \in \pi_i} w(x, y, G) = w(\pi_i, G) \geq d(u, v, G). \end{aligned}$$

To prove Inequality (3), let π be a shortest h -hop path in G . Observe that $w(\pi, G_i) \geq d(u, v, G_i)$, as again no path has smaller weight than the shortest path in G_i . By additionally exploiting the assumption $d^h(u, v, G) \geq 2^i$, we get

$$\begin{aligned} d(u, v, G_i) \cdot \rho_i &\leq w(\pi, G_i) \cdot \rho_i = \sum_{(x,y) \in \pi} w(x, y, G_i) \cdot \rho_i \\ &= \sum_{(x,y) \in \pi} \left\lceil \frac{w(x, y, G)}{\rho_i} \right\rceil \cdot \rho_i \\ &\leq \sum_{(x,y) \in \pi} (w(x, y, G) + \rho_i) \\ &= w(\pi, G) + |\pi| \cdot \rho_i \\ &= d^h(u, v, G) + |\pi| \cdot \rho_i \\ &\leq d^h(u, v, G) + h \cdot \rho_i \\ &= d^h(u, v, G) + \epsilon 2^j \\ &\leq d^h(u, v, G) + \epsilon d^h(u, v, G) \\ &= (1 + \epsilon) d^h(u, v, G). \end{aligned}$$

To prove Inequality (3), we combine Inequality (4) with the assumption $d^h(u, v, G) \leq 2^{i+1}$:

$$d(u, v, G_i) \leq \frac{(1 + \epsilon) d^h(u, v, G)}{\rho_i} = \frac{(1 + \epsilon) h d^h(u, v, G)}{\epsilon 2^j} \leq \frac{(1 + \epsilon) h 2^{j+1}}{\epsilon 2^j} = (1 + 2/\epsilon) h.$$

B Proof of Lemma 3.1

We prove the claim by the probabilistic method. Consider a sampling process that determines a set $T \subseteq U$ by adding each element of U to T independently with probability $1/(2x)$. Let E_0 denote the event that $|T| > |U|/x$ and for every $1 \leq i \leq k$ let E_i denote the event that $T \cap S_i = \emptyset$. First, observe that the size of T is $|U|/(2x)$ in expectation. By Markov's inequality, we can bound the probability that the size of T is at most twice the expectation by at least $1/2$ and thus $\Pr[E_0] = \Pr[|T| > |U|/x] \leq 1/2$. Furthermore, for every $1 \leq i \leq k$, the probability that S_i contains no node of T is

$$\Pr[E_i] = \left(1 - \frac{1}{2x}\right)^{|S_i|} \leq 1 - \left(1 - \frac{1}{2x}\right)^{2x \ln 3k} \leq \frac{1}{e^{\ln 3k}} = \frac{1}{3k}.$$

The set T fails to have the desired properties of a small hitting set if at least one of the events E_i occurs. By the union bound we have

$$\Pr\left[\bigcup_{0 \leq i \leq k} E_i\right] \leq \sum_{0 \leq i \leq k} \Pr[E_i] \leq \frac{1}{2} + k \cdot \frac{1}{3k} = \frac{1}{2} + \frac{1}{3} < 1.$$

It follows that the sampling process constructed a hitting set T for $\mathcal{C} = \{S_1, \dots, S_k\}$ of size at most $|T| \leq |U|/x$ with non-zero probability. Therefore a set T with these properties must really exist. This finishes the proof of Lemma 3.1.

C Ruling Set Algorithm

For each node v , we represent its ID by a binary number $v_1 v_2 \dots v_\lambda$. Initially, we set $T_0 = U$. The algorithm proceeds for b iterations.

In the j^{th} iteration, we consider u_j for every node $u \in T_{j-1}$. If $u_j = 0$, u remains in T_j and sends a “beep” message to every node within distance $c - 1$. This takes $c - 1$ rounds as beep messages from different nodes can be combined. If $u_j = 1$, it remains in T_j if there is no node $v \in T_{j-1}$ such that $d(u, v, G) \leq c$ and $v_j = 0$; in other words, it remains in T_j if it does not hear any beep after $c - 1$ rounds. The output is $T = T_\lambda$. The running time of the above algorithm is clearly $O(c\lambda) = O(c \log n)$. Also, the distance between every pair of nodes in T is at least c since for every pair of nodes u and v of distance less than c , there is a j such that $u_j \neq v_j$, and in the j^{th} iteration if both u and v are in T_{j-1} , then one of them will send a beep and the other one will not be in T_j . Finally, it can be shown by induction that after the j^{th} round every node in U is at distance at most jc from some node in T_j ; thus it follows that every node in U is at distance at most $c\lambda$ from some node in T .